Festschrift

LNCS 7230

Robert L. Constable
Alexandra Silva (Eds.)

Logic and Program Semantics

Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday

Springer

Lecture Notes in Computer Science 7230

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Robert L. Constable Alexandra Silva (Eds.)

Logic and Program Semantics

Essays Dedicated to Dexter Kozen
on the Occasion of His 60th Birthday

Springer

Volume Editors

Robert L. Constable
Cornell University
Computer Science Department
4149 Upson Hall, Ithaca, NY 14853, USA
E-mail: rc@cs.cornell.edu

Alexandra Silva
Radboud University Nijmegen
Intelligent Systems Faculty of Science
Institute for Computing and Information Sciences
Postbus 9010, 6500 GL Nijmegen, The Netherlands
E-mail: alexandra@cs.ru.nl

Cover illustration:
Photographer: Jason Koski (UPHOTO)
© Cornell University Photography
Description: McGraw Tower and Uris Library at sunset in fall.

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-29484-6 e-ISBN 978-3-642-29485-3
DOI 10.1007/978-3-642-29485-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012935014

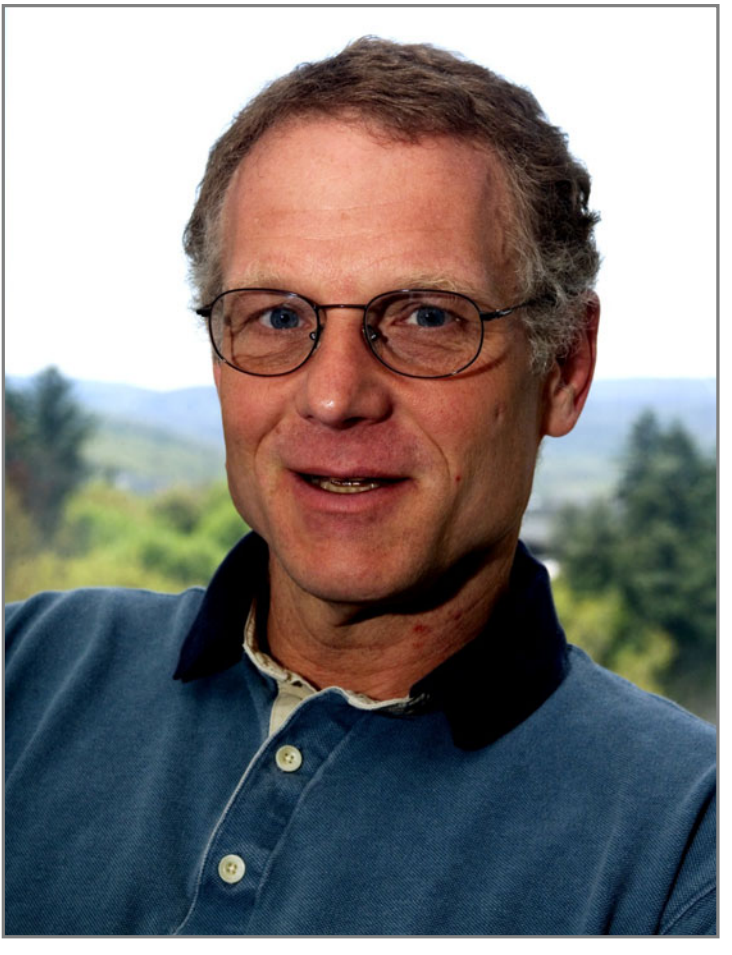
CR Subject Classification (1998): F.3, D.3, D.2, F.4.1, F.1, F.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)



Dexter Kozen

Preface

Dexter Kozen is a personal force in computer science. Even those who have met him only once come away with an almost physical sense of his intellectual horsepower, his boundless energy, and his intellectual depth. The contributors to this volume bear witness to his influence, and their eagerness to join us in this enterprise confirmed our expectations that we would attract a diverse and enthusiastic group of authors, revealing the vast range of Dexter's interests and contributions. One of the two editors has known Dexter longer than the other one has known her parents. Both of them will add their *laudatios* in this Preface. First we introduce the wide spectrum of contributions in this volume that reflect the breadth of Dexter's work and influence.

Dexter has been a leader in the development of Kleene Algebras (KAs), and the article by Andreka, Mikulas, and Nemeti presents a new result on axiomatizing Residuated Kleene Algebras. The paper by Kupke and Rutten looks at a coalgebraic approach to automatic sequences. Dexter was inspired by the work of Rutten and his coauthors to examine a coalgebraic approach to KAs with tests (KATs). His interest in coalgebraic methods attracted the article by Bonchi, Bonsangue, Rutten, and Silva on Brzozowski's minimization algorithm for finite automata. This looks like another algorithm that is ripe for formalization and perhaps for extraction from a formal proof as was done for the minimization algorithm in the classic 1969 textbook of Hopcroft and Ullman, *Formal Languages and Their Relation to Automata* using the Nuprl prover. Dexter's results on congruence closure have been used in the Nuprl system for years, invoked thousands of times a week at Cornell alone. The article by Kreitz discusses other ways in which Nuprl has been a formal partner in Dexter's work, a kind of self application of Kozen to Kozen. Indeed, we thought of writing an article on the formal results in the Nuprl digital library that are related to Dexter's work, but then we saw that these connections would be manifest in this collection. The article by Jeannin on capsules is another example where the elegance of ideas that Dexter develops with his students influence implementation work at Cornell and elsewhere.

The article by Panangaden, Knight, and Mardare on completeness of epistemic logic represents another topic on which Dexter has done influential work, namely, the completeness of various programming logics. In the same topic, the paper by Moss, Wennstrom, and Whitney presents a complete logical system for the equality of recursive terms for sets. This theme is closely related to the theme of finding decision procedures for logics as presented in the article by Rehof and Urzyczyn. This article uses results on alternation, a topic of Dexter's research for which he won the Outstanding Innovation Award from IBM in 1980. Another work related to alternation is the paper by Michalewski and Niwinski. Still related to logic, the volume includes two papers on game semantics. The paper

by Winskel presents a bicategorical formulation of games representing concurrent programs and processes. Parikh, Tasdemir, and Witzel discuss choice and uncertainty in games.

Donald discusses in detail the impact that one of the first papers of Dexter, "On the Power of the Compass," co-authored with Manuel Blum, has had in robotics and nanoscience. This work fits in another field where Dexter has made pioneering contributions, namely, in the area of (algebraic) algorithms and complexity. In the same topic fit the paper by Palsberg, who presents a tutorial proof that overloading is NP-complete, the paper by Carmosino, Immerman, and Jordan, on descriptive complexity, presenting a tool for performing research and learning about finite model theory, and also the papers by Chen and Sharp. Chen's paper discusses the complexity of the quantified constraint satisfaction problem on finite structures and Sharp discusses the complexity of distance coloring in graphs.

It is gratifying to see papers from three of Dexter's graduated PhD students, Hubie Chen, Neal Glew, and Alexa Sharp. We discussed above the papers by Chen and Sharp. Glew's paper, on subtyping and equirecursive types, is a prime example of another area in which Dexter has worked, namely, programming languages and program analysis.

The volume also includes articles not directly related to Dexter's research but which add to the feeling of diversity that has always characterized his science. Salomaa, who investigated in the past one of Dexter's favorite research topics (completeness of KAs), presents a paper where he studies reaction systems useful to model biochemical reactions. Gorecki and Tiuryn, the latter co-author, with Dexter and David Harel, of a book on dynamic logic, present a quite elaborate paper on phylogenetics.

The second part of this volume includes *laudatios* from several collaborators, students and friends, including the members of his current band. The two editors add their *laudatios* below.

The first editor has known Dexter since 1976 when Juris Hartmanis brought Dexter to his office and said something like, "Here is a very clever chap, one of my PhD students, that you should know since he is also interested in the kind of thing you do." I think Juris might have added "strange as that kind of thing is." It didn't take me long to see this truth, and my students and I were implementing his very clever congruence closure algorithm in our PLCV Programming Logic system by 1978. We have kept that algorithm as a part of our interactive provers ever since, modifying it to handle types as our systems evolved. Over the years my students and I have been influenced by countless ideas and insights from Dexter. More broadly, the students at Cornell universally admire Dexter for his exceptionally precise and clear lectures. Their style can be seen in the four textbooks Dexter has written: *The Design and Analysis of Algorithms*, *Automata and Computability*, *Dynamic Logic* (with Harel and Tiuryn), and *Theory of Computation* – all but one published by Springer. I have taught from his unpublished lecture notes as well, and that is a remarkable experience. Everything is perfectly explained. It is no wonder that he is regarded

by so many Cornell students as the best teacher that ever taught them. All of Dexter's colleagues have benefited from his extraordinary teaching ability and from his books and polished lecture notes. Those of us who have played ultimate frisbee, hockey, and tennis with Dexter know that he is also a remarkable athlete who can then step from the hockey rink to the stage and wow an audience with his musical talents. Dexter is one of a kind.

The second editor met Dexter in Amsterdam when she was a first-year insecure PhD student. Talking to Dexter for the first time was a revealing moment, and after just a few minutes into the conversation she realized how much more than only a brilliant researcher Dexter was. His ability to explain a solution to a problem is astonishing and reading his papers was a great learning tool along the years. The months she spent in Ithaca after her PhD were great in many ways, and she is grateful to Dexter and Fran for having made all the efforts to make her, and later Jan Rutten and Marcello Bonsangue, welcome in the Finger Lakes' region. From the many valuable lessons she learned from Dexter, she chooses one to share with the readers: *a beautiful result deserves a beautiful proof*. Dexter puts an amazing energy into everything he does and she is proud to know him and to have been given the opportunity to work with him. She wishes him all the best for the many years to come!

We are grateful to everyone who has participated in putting together this volume and the symposium in honor of Dexter. We thank all the authors for writing wonderful papers that will certainly delight Dexter and also for helping us with the reviewing process. Several people at Cornell, including Michelle Eighmey, Jim Entwood, Tammy Gardner, Joe Halpern and Juris Hartmanis, deserve a special mention for all their efforts in the organization of the symposium. We also thank everyone at Springer, most notably Alfred Hofmann and Anna Kramer, who embraced this project with enthusiasm and helped us in composing the book.

We use the title of one of the laudatios in this volume, by David Harel, to conclude this preface. It has been our greatest pleasure to organize this volume as a tribute to *Dexter Kozen – A Winning Combination of Brilliance, Depth, and Elegance*.

April 2012

Robert L. Constable
Alexandra Silva

Table of Contents

Part I: Full Papers

Part II: Laudatios

Residuated Kleene Algebras*

Hajnal Andréka[1], Szabolcs Mikulás[2], and István Németi[1]

[1] Alfréd Rényi Institute of Mathematics
Hungarian Academy of Sciences
13–15 Reáltanoda u., 1053 Budapest, Hungary
{andreka,nemeti}@renyi.hu
[2] Department of Computer Science and Information Systems
Birkbeck, University of London
Malet Street, London WC1E 7HX, UK
szabolcs@dcs.bbk.ac.uk

Abstract. We show that there is no finitely axiomatizable class of algebras that would serve as an analogue to Kozen's class of Kleene algebras if we include the residuals of composition in the similarity type of relation algebras.

1 Introduction

One of the standard interpretations of Kleene algebras is by families of binary relations: relational Kleene algebras, RKA. It is well known that the equational theory of RKA is not finitely axiomatizable; see [14] or [4, Theorem 9] in English. But Kozen [9] proved that there are intuitive quasi-equations (i.e., equational implications) valid in RKA which together with finitely many equations do axiomatize the equational theory of RKA. The so obtained quasi-variety is Kozen's class of Kleene algebras, KA. Thus there is a finitely axiomatized quasi-variety $\mathsf{KA} \supseteq \mathsf{RKA}$ generating the same variety as RKA, i.e., using the terminology of Definition 2.3,

KA provides a strong, finite quasi-axiomatization of RKA.

Pratt [13] observed that including the residuals of composition into the similarity type of Kleene algebras has the advantage that the resulting class is a finitely axiomatizable variety; with the use of the residuals the quasi-equations in the axiomatization of KA can be expressed as equations. Following Pratt we will call the class of Kleene algebras equipped with the residuals of composition *action algebras*, AA, and we will call the subclass of action algebras that can be interpreted over families of binary relations *relational action algebras*, RAA. Thus, RAA consists of members of RKA equipped with the residuals.

In this paper we prove that the price of including the residuals into the similarity type, and so turning KA into a variety AA, is not only that the equational theory of RAA is not finitely axiomatizable, but there is no strong, finite

* Research supported by the Hungarian National Foundation for Scientific Research grant No. T81188.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 1–11, 2012.

quasi-axiomatization for RAA; see Theorem 3.2. Thus, the "trick" in defining KA cannot be done again for RAA.

Kozen [10] expands the similarity type of action algebras by meet. The resulting class, *action lattices*, AL, is again a finitely axiomatizable variety. We define the class of *relational action lattices*, RAL, as that subclass of AL whose elements can be represented on binary relations. The equational theory of RAL turns out to be nonfinitely axiomatizable; see [7]. Here we give an alternative, simpler proof of this fact, and show that strong, finite quasi-axiomatization is impossible in this case, too; see Theorem 4.1.

The rest of the paper is organized as follows. In the next section we give precise definitions of the classes of algebras to be investigated and a formal definition of finite quasi-axiomatizability. Then we turn our attention to action algebras in Section 3. In Section 4 we look at action lattices. We conclude with some open problems.

2 Basics

Given a similarity type Λ, we denote the class of relational Λ-algebras by $\mathsf{R}(\Lambda)$: the class of those Λ-algebras that are isomorphic to algebras of binary relations where the elements of Λ are interpreted as "natural" operations on binary relations. Using the terminology of the relation algebra literature, we will sometimes refer to elements of $\mathsf{R}(\Lambda)$ as *representable* algebras. Below we give precise definitions of $\mathsf{R}(\Lambda)$ for particular choices of Λ.

Definition 2.1 (Relational Kleene algebras). *The class of* relational Kleene algebras *is*

$$\mathsf{RKA} = \mathsf{R}(+, ;, ^*, 0, 1') \ ,$$

i.e., the class of subalgebras of algebras of the form $(\wp(W), +, ;, ^*, 0, 1')$ *where* W *is an equivalence relation,* $+$ *is set union,* $;$ *is relation composition*

$$x \,;\, y = \{(u, v) \in W : (u, w) \in x \text{ and } (w, v) \in y \text{ for some } w\} \ ,$$

* *is reflexive–transitive closure,* 0 *is the emptyset, and* $1'$ *is the identity relation restricted to* W

$$1' = \{(u, v) \in W : u = v\} \ .$$

We recall the interpretation of the residuals $\setminus$ and $/$ of composition in relation algebras:

$$x \setminus y = \{(u, v) \in W : \forall w((w, u) \in x \text{ implies } (w, v) \in y)\}$$
$$x \,/\, y = \{(u, v) \in W : \forall w((v, w) \in y \text{ implies } (u, w) \in x)\} \ .$$

Next we define expansions of relational Kleene algebras with residuals and meet; see [13,10,8] for similar expansions.

Definition 2.2 (Relational Residuated Kleene Algebras). *The class of* relational action algebras *is defined as*

$$\mathsf{RAA} = \mathsf{R}(+, ;, ^*, /, \backslash, 0, 1')$$

while the class of relational action lattices *is defined as*

$$\mathsf{RAL} = \mathsf{R}(\cdot, +, ;, ^*, /, \backslash, 0, 1')$$

where $\cdot$ *is interpreted as intersection.*

We note that we would get an equivalent definition if we require that W is a Cartesian square $U \times U$ in the above definitions. We chose W be an equivalence relation, since we will sometimes include the top element 1 into Λ, and requiring that 1 is interpreted as a Cartesian square would result in classes that are not closed under products. Other additional operations we will consider in this paper are complement $-$ and converse $^\smile$. See [5,6] for Kleene algebras expanded with converse.

It may be useful to introduce some terminology to describe when the equational theory of a class of algebras has a finite quasi-equational axiomatization.

Definition 2.3. *Given a class* K *of algebras, we say that (the equational theory of)* K *is* finitely quasi-axiomatized *if there is a quasi-variety* Q *such that*

- Q *and* K *generate the same variety, i.e., their equational theories coincide:* $\mathsf{Eq}(\mathsf{Q}) = \mathsf{Eq}(\mathsf{K})$,
- Q *is finitely axiomatizable.*

If, in addition,

- $\mathsf{K} \subseteq \mathsf{Q}$, *i.e., the axioms of* Q *are valid in* K,

then we say that the finite quasi-axiomatization is strong.

3 Action Algebras

In this section we show that relational action algebras do not have a strong, finite quasi-axiomatization; see Theorem 3.2.

It is well known that the classes RAA and RAL are not axiomatizable by first-order logic formulas, because the presence of reflexive–transitive closure in their signatures causes them to be not closed under ultraproducts. The following theorem cited from [2, Theorem 5.1] implies that even their quasi-equational theories are not finitely axiomatizable.

Theorem 3.1. *Let* $\{+, ;\} \subseteq \Lambda \subseteq \{+, ;, ^*, /, \backslash, ^\smile, 0, 1', 1\}$. *Neither* $\mathsf{R}(\Lambda)$ *nor the quasi-equational theory of* $\mathsf{R}(\Lambda)$ *is finitely axiomatizable.*

Now, the equational theory of $\mathsf{R}(+,;)$ is different, it is finitely axiomatizable (by equations), and moreover the equational theories of $\mathsf{R}(\Lambda)$ where the residuals and transitive closure are not included in Λ tend to be finitely axiomatizable. For a complete description of the cases see [2]. Our next theorem shows that this situation changes radically if we include the residuals into the similarity types.

Theorem 3.2. *Let* $\{+,;,/,\backslash\} \subseteq \Lambda \subseteq \{+,;,^*,/,\backslash,^\smile,0,1',1\}$. *The equational theory of* $\mathsf{R}(\Lambda)$ *is not finitely axiomatizable.*

Moreover, there is no strong, finite quasi-axiomatization of $\mathsf{R}(\Lambda)$. *In fact, there is no first-order logic formula valid in* $\mathsf{R}(+,;,^*,/,\backslash,^\smile,0,1',1)$ *which implies all the equations valid in* $\mathsf{R}(+,;,/,\backslash)$.

Proof. We recall, for every natural number n, the algebra

$$\mathfrak{A}_n = (A_n, +, ;, ^*, /, \backslash, ^\smile, 0, 1', 1)$$

from [2, Theorem 5.1].

We define

$$G_n = \{a, a'_1, a''_1, \ldots, a'_n, a''_n, b, b'_1, b''_1, \ldots, b'_n, b''_n, o, 1', 0\} .$$

Let $(A_n, +)$ be the free upper semilattice generated freely by G_n under the defining relations:

$$\{a \le a'_i + a''_i,\ b \le b'_i + b''_i,\ 0 + x = x : 1 \le i \le n, x \in G_n\} .$$

Let S denote the following set of two-element subsets of A_n:

$$S = \big\{\{a, b'_1\}\big\} \cup \big\{\{a'_i, b''_i\} : 1 \le i \le n\big\} \cup \big\{\{a''_i, b'_{i+1}\} : 1 \le i < n\big\} \cup \big\{\{a''_n, b\}\big\} .$$

Next we define the rest of the operations on A_n as follows:

$$0 = \emptyset \qquad 1 = \sum G_n \qquad x^\smile = x$$
$$0 ; x = 0 = x ; 0 \qquad 1' ; x = x = x ; 1'$$
$$\text{if } x, y \notin \{0, 1'\}, \text{ then } x ; y = \begin{cases} o & \text{if } \{x, y\} \in S \\ 1 & \text{otherwise.} \end{cases}$$

Kleene star * is defined as follows. We have $0 ; 0 = 0$, $1' ; 1' = 1'$ and $x ; x = 1$ for every $x \in A_n \setminus \{0, 1'\}$. Hence we define, in $\mathfrak{A}_n$, $0^* = 1'$, $1'^* = 1'$ and $x^* = 1$ for every $x \in A_n \setminus \{0, 1'\}$.

We define the residual $\backslash$ in the algebras $\mathfrak{A}_n$ so that $x \setminus y$ is the largest element z such that $x ; z \le y$. Then the algebras $\mathfrak{A}_n$ are in fact closed under the operation $\backslash$ (since they are finite). Indeed, the extension of $x \setminus y$ is determined by

$$z \le x \setminus y \text{ iff } x ; z \le y .$$

Note that this defines $/$ as well, since $/$ and $\backslash$ coincide in symmetric algebras (where $x ; y = y ; x$ is valid).

It is shown in [2] that

1. the $\{+,;\}$-reduct of $\mathfrak{A}_n$ is not representable,
2. any nontrivial ultraproduct over ω, $\mathfrak{A}$, of the $\mathfrak{A}_n$'s is representable.

Item 1 above was shown by constructing a quasi-equation q_n for every n as

$$\bigwedge_{i=1}^{n}(x \leq x_i' + x_i'' \wedge y \leq y_i' + y_i'') \rightarrow$$
$$x\,;y \leq x\,;y_1' + \sum_{i=1}^{n-1}(x_i'\,;y_i'' + x_i''\,;y_{i+1}') + x_n'\,;y_n'' + x_n''\,;y\ .$$

By an induction on n one can show that q_n is valid in representable algebras. On the other hand, the evaluation ϵ given by

$$\epsilon(x) = a \quad \epsilon(x_i') = a_i' \quad \epsilon(x_i'') = a_i'' \quad \epsilon(y) = b \quad \epsilon(y_i') = b_i' \quad \epsilon(y_i'') = b_i''$$

falsifies q_n in $\mathfrak{A}_n$ (since $a;b = 1$ and each term on the right of $\leq$ in the consequent evaluates to o).

Here we modify q_n to equation e_n with the same properties, but we have to pay the price of including the residuals into the language. Then it follows that the $\{+,;,/,\backslash\}$-reduct of $\mathfrak{A}_n$ is not in the variety generated by the representable algebras.

Below we will use the following abbreviations $x_i := x_i' + x_i''$ and $y_i := y_i' + y_i''$ for $1 \leq i \leq n$. We define, inductively,

$$\kappa_1 = x \setminus x_1 \qquad \kappa_{i+1} = (x\,;\kappa_1\,;\ldots;\kappa_i) \setminus x_{i+1}$$

and

$$\lambda_1 = y_n \,/\, y \qquad \lambda_{i+1} = y_{n-i+1} \,/\, (\lambda_i\,;\ldots;\lambda_1\,;y)\ .$$

Let τ_n be the term

$$x\,;\kappa_1\,;\ldots;\kappa_n\,;\lambda_n\,;\ldots;\lambda_1\,;y\ .$$

We define σ_n as

$$x\,;\kappa_1\,;\ldots;\kappa_n\,;y_1' + x_1'\,;\kappa_2\,;\ldots;\kappa_n\,;y_1'' + x_1''\,;\kappa_2\,;\ldots;\kappa_n\,;\lambda_n\,;y_2' + \ldots +$$
$$x_{n-1}''\,;\kappa_n\,;\lambda_n\,;\ldots;\lambda_2\,;y_n' + x_n'\,;\lambda_n\,;\ldots;\lambda_2\,;y_n'' + x_n''\,;\lambda_n\,;\ldots;\lambda_1\,;y\ .$$

Finally, e_n is defined as $\tau_n \leq \sigma_n$.

It is not difficult to show that e_n is valid in representable algebras. As an example we show the case $n = 2$. See Figure 1.

Equation e_2 has the form $\tau_2 \leq \sigma_2$ where

$$\tau_2 = x\,;x \setminus x_1\,;(x\,;x \setminus x_1) \setminus x_2\,;y_1 \,/\, (y_2 \,/\, y\,;y)\,;y_2 \,/\, y\,;y$$

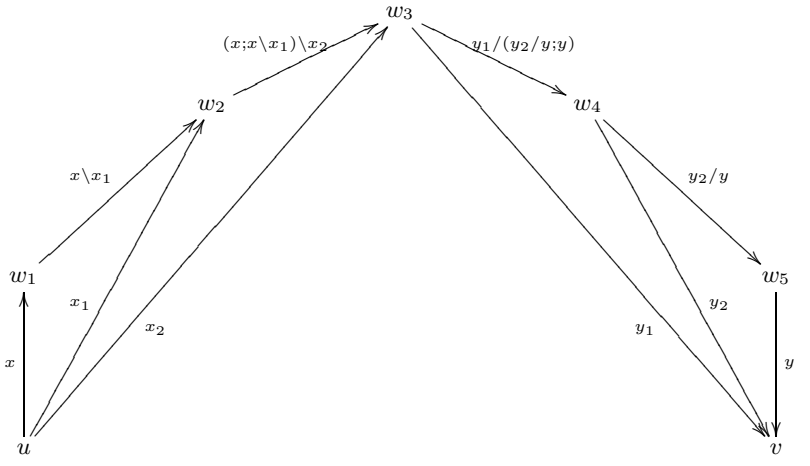


Fig. 1. Validity of e_2

and

$$\begin{aligned}\sigma_2 = {} & x\,;x\setminus x_1\,;(x\,;x\setminus x_1)\setminus x_2\,;y_1'+\\ & x_1'\,;(x\,;x\setminus x_1)\setminus x_2\,;y_1''+\\ & x_1''\,;(x\,;x\setminus x_1)\setminus x_2\,;y_1/(y_2/y\,;y)\,;y_2'+\\ & x_2'\,;y_1/(y_2/y\,;y)\,;y_2''+\\ & x_2''\,;y_1/(y_2/y\,;y)\,;y_2/y\,;y\,.\end{aligned}$$

Let $\mathfrak{A}$ be a representable algebra and assume that $(u,v)\in\tau_2$ in $\mathfrak{A}$ under some valuation (for the sake of simplicity we denote the value of a term as the term itself). Then there are $w_1,w_2,\ldots,w_5$ such that $(u,w_1)\in x$, $(w_1,w_2)\in x\setminus x_1$, $(w_2,w_3)\in (x\,;x\setminus x_1)\setminus x_2$, $(w_3,w_4)\in y_1/(y_2/y\,;y)$, $(w_4,w_5)\in y_2/y$ and $(w_5,v)\in y$. By the definition of composition we get that $(u,w_2)\in x\,;x\setminus x_1$ and $(w_4,v)\in y_2/y\,;y$. Then by the definition of the residuals we have that $(u,w_2)\in x_1=x_1'+x_1''$, $(u,w_3)\in x_2=x_2'+x_2''$, $(w_4,v)\in y_2=y_2'+y_2''$ and $(w_3,v)\in y_1=y_1'+y_1''$.

For a contradiction assume that $(u,v)\notin\sigma_2$. Then $(w_3,v)\notin y_1'$, otherwise we would have $(u,v)\in x;x\backslash x_1;(x;x\backslash x_1)\backslash x_2;y_1'$. Hence $(w_3,v)\in y_1''$. Then $(u,w_2)\notin x_1'$, otherwise we get $(u,v)\in x_1'\,;(x\,;x\setminus x_1)\setminus x_2\,;y_1''$. Hence $(u,w_2)\in x_1''$. Then $(w_4,v)\notin y_2'$, otherwise we get $(u,v)\in x_1''\,;(x\,;x\setminus x_1)\setminus x_2\,;y_1/(y_2/y\,;y)\,;y_2'$. Hence $(w_4,v)\in y_2''$. Then $(u,w_3)\notin x_2'$, otherwise we get $(u,v)\in x_2'\,;y_1/(y_2/y\,;y)\,;y_2''$. But then $(u,w_3)\in x_2''$, whence $(u,v)\in x_2''\,;y_1/(y_2/y\,;y)\,;y_2/y\,;y$. That is, $(u,v)\in\sigma_2$ contrary to the assumption.

On the other hand, the evaluation ϵ given by

$$\epsilon(x)=a\quad\epsilon(x_i')=a_i'\quad\epsilon(x_i'')=a_i''\quad\epsilon(y)=b\quad\epsilon(y_i')=b_i'\quad\epsilon(y_i'')=b_i''$$

falsifies e_n in $\mathfrak{A}_n$. Indeed, $a \setminus a_i$ and b_i / b equal the identity $1'$ for all $1 \leq i \leq n$ (since $a \leq a_i = a_i' + a_i''$ and there is no nonzero element z other than the identity such that $a ; z \leq a_i$). Since $z = z ; 1'$, we get that every term between x and y in τ_n evaluates to $1'$. Hence τ_n evaluates to $a ; b = 1$, By the same reasoning we get that every element of the sum in σ_n evaluates to o. Thus $\epsilon(\tau_n) = 1 \not\leq o = \epsilon(\sigma_n)$.

To finish the proof, let φ be an arbitrary first-order logic formula valid in $\mathsf{R}(+, ;, ^*, /, \setminus, ^\smile, 0, 1', 1)$. Then φ is valid in $\mathfrak{A}$, because the latter is representable. Since $\mathfrak{A}$ is an ultraproduct of the $\mathfrak{A}_n$'s, there is an n such that φ is valid in $\mathfrak{A}_n$. Since the equation e_n is not valid in $\mathfrak{A}_n$, we have that φ does not imply e_n, though the latter is an equation valid in $\mathsf{R}(+, ;, /, \setminus)$. □

As a corollary we get the following.

Theorem 3.3. *The equational theory of relational action algebras is not finitely axiomatizable over the equational theory of relational Kleene algebras, i.e., there is no finite set of* $\mathsf{Eq}(\mathsf{RAA})$ *which together with* $\mathsf{Eq}(\mathsf{RKA})$ *would imply* $\mathsf{Eq}(\mathsf{RAA})$.

Proof. Assume the contrary, i.e., that there is a finite set E which would axiomatize the equational theory $\mathsf{Eq}(\mathsf{RAA})$ of RAA over the equational theory $\mathsf{Eq}(\mathsf{RKA})$ of RKA. Since the finite set E' of equational axioms of AA in [13] implies $\mathsf{Eq}(\mathsf{RKA})$, then the finite set $E \cup E'$ would axiomatize $\mathsf{Eq}(\mathsf{RAA})$, which is impossible according to Theorem 3.2. □

4 Action Lattices

In this section we look at similarity types that include the meet operation as well. Since Theorem 3.2 does not apply to signatures that include meet, we need another construction to show nonfinite axiomatizability of the equational theory.

The following result follows from [7, Theorem 2.3 and Corollary 4.4.]. The construction in [7] is rather involved, since we needed dense algebras (where $x \leq x;x$ is valid) so that they can be applied to relevance logic. A simpler construction is available from [11] that has been used to show nonfinite axiomatizability of the equational theories of some residuated algebras in [12]. The Kleene star operation was not considered in [12], but it can be easily added to the signature. Also, complementation $-$ can be included in the signature; in the representable algebras $-$ is interpreted as complementation with respect to the largest element of the algebra.

Theorem 4.1. *Let* $\{\cdot, +, ;, \setminus\} \subseteq \Lambda \subseteq \{\cdot, +, -, ;, ^*, /, \setminus, ^\smile, 0, 1', 1\}$. *The equational theory of* $\mathsf{R}(\Lambda)$ *is not finitely axiomatizable.*

Moreover, there is no strong, finite quasi-axiomatization of $\mathsf{R}(\Lambda)$. *In fact, there is no first-order logic formula valid in* $\mathsf{R}(\cdot, +, -, ;, ^*, /, \setminus, ^\smile, 0, 1', 1)$ *which implies all the equations valid in* $\mathsf{R}(\cdot, +, ;, \setminus)$.

Proof. We recall the main features of the algebras $\mathfrak{A}_n = (A_n, -, \cdot, +, ;, ^\smile, 0, 1', 1)$ of [11]. $\mathfrak{A}_n$ has the following atoms (minimal, nonzero elements): identity $1'$, q_i

for $1 \leq i \leq m$, and p_j for $1 \leq j \leq n$ with $m = 3 \cdot n!$. Every atom is self converse: $x^\smile = x$. Composition is defined so that

$$q_{i+1} \leq p_1 \,;\, q_i \qquad \text{for every } 1 \leq i < m \tag{1}$$

$$0 = q_r \cdot q_s \,;\, q_t \qquad \text{for every } 1 \leq r, s, t \leq m \tag{2}$$

$$0 = p_l \cdot p_l \,;\, p_l \qquad \text{for every } 1 \leq l \leq n. \tag{3}$$

The elements of $\mathfrak{A}_n$ are the subsets of the atoms, and $+, \cdot, -$ are defined as the corresponding set theoretic operations. Composition ; and converse $^\smile$ distribute over $+$. We define $x \setminus y := -(x^\smile \,;\, -y)$ and $x \,/\, y := -(x \,;\, y^\smile)$. It follows that (the $\{\cdot, +, ;\}$-reduct of) $\mathfrak{A}_n$ is not representable (since that would imply the existence of a colouring of the edges of a total graph of m vertices with n colours without monochromatic triangles; an impossible task). See Figure 2, where every dotted arrow should have a color p_i for some $1 \leq i \leq n$, and [1,12] for further details.

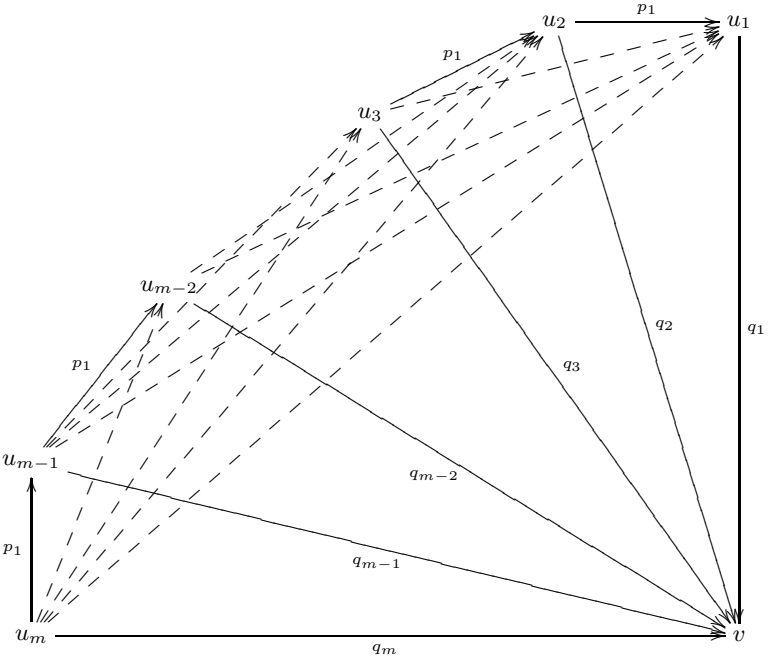


Fig. 2. The reason for non-representability

Furthermore, for every nonidentity atom x, we have that $x^3 := x \,;\, x \,;\, x = 1$ in $\mathfrak{A}_n$. Thus we can define Kleene star in $\mathfrak{A}_n$ as

$$x^* = \begin{cases} 1' & x \in \{0, 1'\} \\ 1 & \text{otherwise.} \end{cases}$$

It is shown in [11] that any nonprincipal ultraproduct $\mathfrak{A}$ of the $\mathfrak{A}_n$'s is representable. By the above definition of Kleene star in $\mathfrak{A}_n$, it follows that we can define * in the same way in $\mathfrak{A}$ and that the representation respects * as well.

It is shown in [12] that there are equations e_n in the language $\{\cdot, +, ;, \backslash\}$ such that they witness the nonrepresentability of $\mathfrak{A}_n$:

- e_n fails in the $\{\cdot, +, ;, \backslash\}$-reduct of $\mathfrak{A}_n$,
- e_n is valid in representable algebras.

We give some intuition on the construction of e_n. To begin with, it is not difficult to define a quasi-equation e'_n expressing that, given the definition of composition in (1) and that the composition p_1^i of i many p_1 is below $\sum_j p_j + \sum_k q_k$, either (3) (there are no monochromatic p-triangles) or (2) (there are no q-triangles) is violated, or some of the q-atoms coincide. For instance, we can define e'_n as

$$\bigwedge_i y_{i+1} \leq x_1 \,;\, y_i \wedge \bigwedge_i x_1^i \leq \sum_l x_l + \sum_j y_j \rightarrow$$

$$y_m \leq \sum_{i,j,l} x_1^i \,;\, (x_l \cdot x_l \,;\, x_l) \,;\, y_j + \sum_{i,j,k} x_1^i \,;\, (y_{m-i+1} \cdot y_j \,;\, y_k) + \sum_{i,j\neq k} x_1^i \,;\, (y_j \cdot y_k)$$

where i, j, k range between 1 and m and l ranges between 1 and n. By the definition of composition, e'_n fails in $\mathfrak{A}_n$ (the antecedent is true, but every element of the sums in the consequent evaluates to 0 when we evaluate x_l to p_l and y_i to q_i). On the other hand, in representable algebras, assuming the antecedent, every y_m-edge in the representation has a decomposition indicated by one of the elements in the sums (otherwise the representation would yield a graph colouring without monochromatic triangles). Finally, using the expressive power of the residuals, e'_n can be equivalently reformulated as an equation e_n — for the technical details we refer the interested reader to [12].

Hence there are algebras, the Λ-reducts of $\mathfrak{A}_n$, which are not in the variety generated by $\mathsf{R}(\Lambda)$ but their nonprincipal ultraproducts are in this variety. From here on the proof ends as in that of Theorem 3.2. □

Similarly to action algebras we get the following corollary.

Corollary 4.2. *The equational theory of relational action lattices is not finitely axiomatizable over the equational theory of relational Kleene algebras, i.e., there is no finite set of* Eq(RAL) *which together with* Eq(RKA) *would imply* Eq(RAL).

5 Conclusion and Open Problems

We mentioned in the introduction that KA provides a *strong*, finite quasi-axiomatization of RKA. That is, the equational theory of RKA follows from a finite conjunction φ of quasi-equations *valid in* RKA; see Kozen [9] for this fact and [6] for references to related results. We proved that the analogous result does not hold for representable residuated algebras, even if we allow φ to be any first-order logic formula valid in representable algebras. Our first question is whether

we can relax the requirement that φ be valid in representable algebras; and if we relax this condition whether we can choose φ to be a conjunction of quasi-equations. This would mean to find a finitely axiomatized quasi-variety Q which generates the same variety as the representable algebras (but we do not require $\mathsf{RAA} \subseteq \mathsf{Q}$ or $\mathsf{RAL} \subseteq \mathsf{Q}$). For motivation we mention that such a quasi-variety would provide us with a finite quasi-equational axiomatization from which we could derive precisely those equations which are valid in representable algebras. Thus, in this respect, validity of the axioms that are not equations is not crucial.

Problem 5.1. Do RAA and RAL have finite quasi-axiomatizations? That is, are there finitely axiomatized quasi-varieties Q_1 and Q_2 such that they generate the same varieties as generated by RAA and RAL, respectively?

By Kozen's result in [9] and our Theorem 3.1 we get, with an argument analogous to the proof of Theorem 3.3, that the quasi-equational theory of RKA is not finitely axiomatizable over the equational theory of RKA. In view of our Theorem 3.2, the same argument does not work for RAA, i.e., it is possible perhaps that the quasi-equational theory of RAA is finitely axiomatizable over the equational theory of RAA. We can hope even for the two theories to coincide in this case, because of the following. As Pratt's work shows, the presence of the residuals of composition allows us to express certain quasi-equations as equations. We are not aware of any general result that would characterize precisely when this can be done. Perhaps this can be done for sufficiently many quasi-equations. Hence we formulate the following question.

Problem 5.2. Do the quasi-variety and the variety generated by RAA (and by RAL) coincide?

Finally, we mention the same finite axiomatizability problem we addressed in this paper in the context of language algebras. It is well known that the equational theories of relational and language Kleene algebras coincide. However, their quasi-equational theories already differ, there are fewer language Kleene algebras than relational ones. Also the equational theories differ if we include meet into the similarity type; see [3] on the connections between the relational and the language models of Kleene algebras/lattices.

Problem 5.3. Are the equational theories of action algebras and action lattices interpreted over languages finitely axiomatizable? If the answer is negative, do they have (strong) finite quasi-axiomatizations?

In connection with this problem we mention that the algebras we used to prove nonfinite axiomatizability in the relational case are of limited use, since their ultraproduct is not representable as a language algebra. The reason is that it contains elements x that are disjoint from the identity $1'$, yet $x ; x \geq 1'$. This cannot happen in a language algebra, since $1'$ is represented as the singleton language containing only the empty word.

Added in proof: During proof-reading this paper we came across the article Buszkowski, W.: On the complexity of the equational theory of relational action algebras. In Schmidt, R.A. (ed.) Relations and Kleene Algebra in Computer Science. LNCS vol. 4136, pp. 106–119. Springer (2006), proving that the complexity of the equational theory of relational residuated Keene algebras is so high (Π^0_1-hard) that it is impossible to give a recursive axiomatization. This gives negative answers to the problems in Section 5 when the Kleene star * is included in the similarity type. In the light of Buszkowski's paper, our theorems say that * is not solely responsible for the nonfinite quasi-axiomatizability: already the *-free equations are so complicated that not even the strong * can help finitely axiomatize them.

References

1. Andréka, H.: Representation of distributive lattice-ordered semigroups with binary relations. Algebra Universalis 28, 12–25 (1991)
2. Andréka, H., Mikulás, S.: Axiomatizability of positive algebras of binary relations. Algebra Universalis 66(1), 7–34 (2011)
3. Andréka, H., Mikulás, S., Németi, I.: The equational theory of Kleene lattices. Theoretical Computer Science 412, 7099–7108 (2011)
4. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
5. Crvenković, S., Dolinka, I., Ésik, Z.: The variety of Kleene algebras with conversion is not finitely based. Theoretical Computer Science 230, 235–245 (2000)
6. Ésik, Z., Bernátsky, L.: Equational properties of Kleene algebras of relations with conversion. Theoretical Computer Science 137, 237–251 (1995)
7. Hirsch, R., Mikulás, S.: Positive reducts of relevance logic and algebras of binary relations. Review of Symbolic Logic 4(1), 81–105 (2011)
8. Jipsen, P.: From semirings to residuated Kleene lattices. Studia Logica 76(2), 291–303 (2004)
9. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110, 366–390 (1994)
10. Kozen, D.: On action algebras. In: van Eijck, J., Visser, A. (eds.) Logic and Information Flow, pp. 78–88. MIT Press (1994)
11. Maddux, R.D.: Non-finite-axiomatizability results for cylindric and relation algebras. Journal of Symbolic Logic 54(3), 951–974 (1989)
12. Mikulás, S.: On representable ordered residuated semigroups. Logic Journal of the IGPL 19(1), 233–240 (2011)
13. Pratt, V.: Action Logic and Pure Induction. In: van Eijck, J. (ed.) JELIA 1990. LNCS, vol. 478, pp. 97–120. Springer, Heidelberg (1991)
14. Redko, V.N.: On defining relations for the algebra of regular events. Ukrain. Mat. Z. 16, 120–126 (1964) (in Russian)

Brzozowski's Algorithm (Co)Algebraically

Filippo Bonchi[1], Marcello M. Bonsangue[2,3],
Jan J.M.M. Rutten[3,4], and Alexandra Silva[3,4,5]

[1] CNRS, ENS Lyon, Université de Lyon LIP (UMR 5668)
[2] LIACS - Leiden University
[3] Radboud University Nijmegen
[4] Centrum Wiskunde & Informatica
[5] HASLab / INESC TEC, Universidade do Minho

Abstract. We give a new presentation of Brzozowski's algorithm to minimize finite automata, using elementary facts from universal algebra and coalgebra, and building on earlier work by Arbib and Manes on the duality between reachability and observability. This leads to a simple proof of its correctness and opens the door to further generalizations.

This paper is dedicated to Dexter Kozen on the occasion of his 60th birthday. Dexter always seeks simplicity and crystal-clear proofs in his research: "a beautiful result deserves a beautiful proof" could be the motto of his work. This paper is a tribute to that[⋆].

1 Introduction

Brzozowski's algorithm [6] is a somewhat unusual recipe for minimizing finite state automata: starting with a (possibly non-deterministic) automaton, one reverses its transitions, makes it deterministic, takes the part that is reachable, and then repeats all of this once more. The result will be a deterministic automaton that is the minimization of the original one.

Though an elementary description and correctness proof of the algorithm is not very difficult (see for instance [13, Cor. 3.14]), the algorithm comes to most as a bit of a surprise. Here we try to add to its understanding by presenting a proof that is based on a result by Arbib and Manes [2,3] on the duality between *reachability* and *observability* (the latter is another word for minimality).

We will first present a reformulation of Arbib and Manes' duality result in terms of a bit of elementary algebra and coalgebra. These are the natural mathematical settings for the modelling of reachability and observability, respectively. Ultimately, their duality is due to the fact that the transitions of an automaton X (with input alphabet A) can be modelled both algebraically, as a function of type $X \times A \to X$ and coalgebraically, as a function of type $X \to X^A$. Next, we will derive (the correctness of) Brzozowski's algorithm as a corollary from this duality.

⋆ We are certain that Dexter will try to simplify our proofs and knowing him he will not stop until he does so! We hope we have made it challenging enough!

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 12–23, 2012.

Our reasons for giving this new formulation of Brzozowski's algorithm are the following.

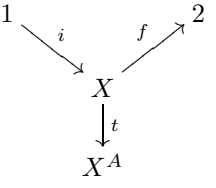
First, the duality between reachability and observability, on which we will base our proof, is in itself a very beautiful result that, unfortunately, it is not very well-known. The original proof by Arbib and Manes uses some category theory that makes it difficult to understand to many. Although our proof of this duality is in essence categorical as well, we have formulated it in elementary terms, using only the notions of sets and functions. As a result, the present paper should be understandable to anyone with a very basic understanding of automata.

Secondly, Brzozowski's algorithm follows as an immediate corollary of this (newly formulated) duality result. This observation gives a new way of understanding the algorithm and makes a formal proof of its correction very easy.

Thirdly, we expect that our proof of Brozowski's algorithm is easy to generalise. The present paper contains the straightforward generalisation of the algorithm to Moore automata. We mention further applications, to weighted and to probabilistic automata, as future work.

2 Rechability and Observability

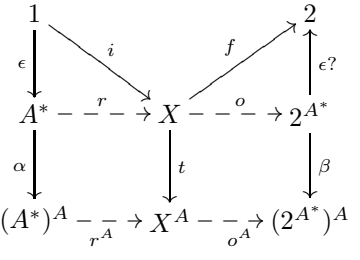
Let $1 = \{0\}$, $2 = \{0, 1\}$ and let A be any set. A deterministic automaton with inputs from A is given by the following data:



(1)

That is: a set X of states; a transition function $t : X \to X^A$ mapping each state $x \in X$ to a function $t(x) : A \to X$ that sends an input symbol $a \in A$ to a state $t(x)(a)$; an initial state $i \in X$ (formally denoted by a function $i : 1 \to X$); and a set of final (or accepting) states given by a function $f : X \to 2$, sending a state to 1 if it is final and to 0 if it is not.

We introduce *reachability* and *observability* of deterministic automata by means of the following diagram:



(2)

in the middle of which we have our automaton X.

On the left, we have the set A^* of all words over A, with the empty word ϵ as initial state and with transition function

$$\alpha : A^* \to (A^*)^A \qquad \alpha(w)(a) = w \cdot a$$

On the right, we have the set 2^{A^*} of all languages over A, with transition function

$$\beta : 2^{A^*} \to (2^{A^*})^A \qquad \beta(L)(a) = \{w \in A^* \mid a \cdot w \in L\}$$

and a final state function

$$\epsilon? : 2^{A^*} \to 2$$

that maps a language to 1 if it contains the empty word, and to 0 if it does not.

Horizontally, we have functions r and o that we will introduce next. First we define x_w, for $x \in X$ and $w \in A^*$, inductively by

$$x_\epsilon = x \qquad x_{w \cdot a} = t(x_w)(a)$$

i.e., x_w is the state reached from x by inputting (all the letters of) the word w. With this notation, we now define

$$r : A^* \to X \qquad r(w) = i_w$$

and

$$o : X \to 2^{A^*} \qquad o(x)(w) = f(x_w)$$

Thus r sends a word w to the state i_w that is reached from the initial state $i \in X$ by inputting the word w. And o sends a state x to the language it accepts. That is, switching freely between languages as maps and languages as subsets,

$$o(x) = \{w \in A^* \mid f(x_w) = 1\} \tag{3}$$

We think of $o(x)$ as the semantics or the *behavior* of the state x.

The functions r and o are homomorphisms in the precise sense that they make the triangles and squares of diagram (2) commute. In order to understand the latter, we note that at the bottom of the diagram, we use, for $f : V \to W$, the notation

$$f^A : V^A \to W^A$$

to denote the function defined by $f^A(\phi)(a) = f(\phi(a))$, for $\phi : A \to V$ and $a \in A$.

One can readily see that the function r is uniquely determined by the functions i and t; similarly, the function o is uniquely determined by the functions t and f. In categorical terms, the unique existence of r is a consequence of A^* being an initial algebra of the functor $1 + (A \times -)$; similarly, the unique existence of o rests on the fact that 2^{A^*} is a final coalgebra of the functor $2 \times (-)^A$.

Having explained diagram (2), we can now give the following definition.

Definition 1 (Reachability and Observability). *A deterministic automaton X is* reachable *if r is surjective. It is* observable *if o is injective.*

Thus X is reachable if all states are reachable from the initial state: for every $x \in X$ there exists a word $w \in A^*$ such that $i_w = x$. And X is observable if different states recognize different languages or, in other words, if they have different observable behavior. We note that an observable automaton is also *minimal*: it does not contain any pair of (language) equivalent states. In what follows, we shall therefore use the words observable and minimal as synonyms.

3 Constructing the Reverse of an Automaton

Next we show that by reversing the transitions, and by swapping the initial and final states of a deterministic automaton, one obtains a new automaton accepting the reversed language. By construction, this automaton will again be deterministic. Moreover, if the original automaton is reachable, the resulting one is minimal.

Our construction will make use the following operation:

$$2^{(-)}: \quad \begin{array}{c} V \\ \downarrow{\scriptstyle f} \\ W \end{array} \quad \mapsto \quad \begin{array}{c} 2^V \\ \uparrow{\scriptstyle 2^f} \\ 2^W \end{array}$$

which is defined, for a set V, by $2^V = \{S \mid S \subseteq V\}$ and, for $f\colon V \to W$ and $S \subseteq W$, by

$$2^f : 2^W \to 2^V \qquad 2^f(S) = \{v \in V \mid f(v) \in S\}$$

(In categorical terms, this is the contravariant powerset functor.)

The main construction: Given the transition function $t\colon X \to X^A$ of our deterministic automaton, we apply, from left to right, the following three transformations:

$$\begin{array}{c} X \\ \downarrow{\scriptstyle t} \\ X^A \end{array} \quad \Vert \quad \begin{array}{c} X \times A \\ \downarrow \\ X \end{array} \quad \vert \quad \begin{array}{c} 2^{X\times A} \\ \uparrow \\ 2^X \end{array} \quad \Vert \quad \begin{array}{c} (2^X)^A \\ \uparrow{\scriptstyle 2^t} \\ 2^X \end{array}$$

The single, vertical line in the middle corresponds to an application of the operation $2^{(-)}$ introduced above. The double lines, on the left and on the right, indicate isomorphisms that are based on the operations of *currying* and *uncurrying*. The end result consists of a new set of states: 2^X together with a new transition function

$$2^t : \ 2^X \to (2^X)^A \qquad 2^t(S)(a) = \{x \in X \mid t(x)(a) \in S\}$$

which maps any subset $S \subseteq X$, for any $a \in A$, to the set of all its a-predecessors. Note that our construction does two things at the same time: it reverses the transitions (as we shall see formally later) and yields again a deterministic automaton.

Initial becomes final: Applying the operation $2^{(-)}$ to the initial state (function) of our automaton X gives

$$\begin{array}{c|c} 1 & 2 \\ \downarrow{\scriptstyle i} & \uparrow{\scriptstyle 2^i} \\ X & 2^X \end{array}$$

(where we write 2 for 2^1), by which we have transformed the initial state i into a final state function 2^i for the new automaton 2^X. We note that according to this new function 2^i, a subset $S \subseteq X$ is final (that is, is mapped to 1) precisely when $i \in S$.

Reachable becomes observable: Next we apply the above construction(s) to the entire left hand-side of diagram (2), that is, to both t and i and to α and ϵ, as well as to the functions r and r^A. This yields the following commuting diagram:

$$\begin{array}{ccc} & & 2 \\ & {\scriptstyle 2^i}\nearrow & \uparrow{\scriptstyle 2^\epsilon} \\ 2^X & \xrightarrow{2^r} & 2^{A^*} \\ \downarrow{\scriptstyle 2^t} & & \downarrow{\scriptstyle 2^\alpha} \\ (2^X)^A & \xrightarrow[2^{r^A}]{} & (2^{A^*})^A \end{array} \tag{4}$$

We note that for any language $L \in 2^{A^*}$, we have $2^\epsilon(L) = \epsilon?(L)$ and, for any $a \in A$,

$$2^\alpha(L)(a) = \{w \in A^* \mid w \cdot a \in L\}$$

The latter resembles the definition of $\beta(L)(a)$ but it is different in that it uses $w \cdot a$ instead of $a \cdot w$. By the universal property (of finality) of the triple $(2^{A^*}, \beta, \epsilon?)$, there exists a unique homomorphism

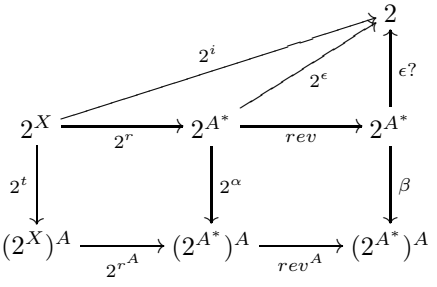
$$\begin{array}{ccc} & & 2 \\ & {\scriptstyle 2^\epsilon}\nearrow & \uparrow{\scriptstyle \epsilon?} \\ 2^{A^*} & \xrightarrow{rev} & 2^{A^*} \\ \downarrow{\scriptstyle 2^\alpha} & & \downarrow{\scriptstyle \beta} \\ (2^{A^*})^A & \xrightarrow[rev^A]{} & (2^{A^*})^A \end{array} \tag{5}$$

which sends a language L to its reverse

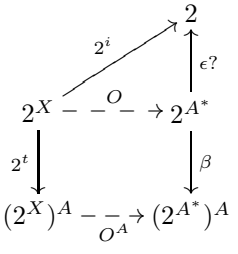
$$rev(L) = \{w \in A^* \mid w^R \in L\}$$

where w^R is the reverse of w.

Combining diagrams (4) and (5) yields the following commuting diagram:



Thus we see that the composition of rev and 2^r (is the unique function that) makes the following diagram commute:

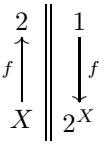


$$O = rev \circ 2^r \tag{6}$$

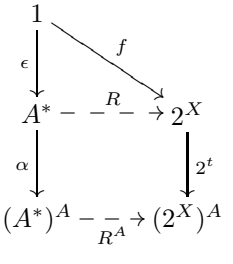
One can easily show that it satisfies, for any $S \subseteq X$,

$$O(S) = \{w^R \in A^* \mid i_w \in S\} \tag{7}$$

Final becomes initial: The following bijective correspondence

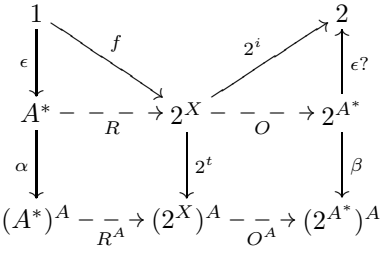


(again an instance of currying) transforms the final state function f of the original automaton X into an initial state function of our new automaton 2^X, which we denote again by f. It will induce, by the universal property of (A^*, ϵ, α), a unique homomorphism as follows:



(8)

Putting everything together: By now, we have obtained the following, new deterministic automaton:



(9)

where the above diagram is simply the combination of diagrams (8) and (6) above.

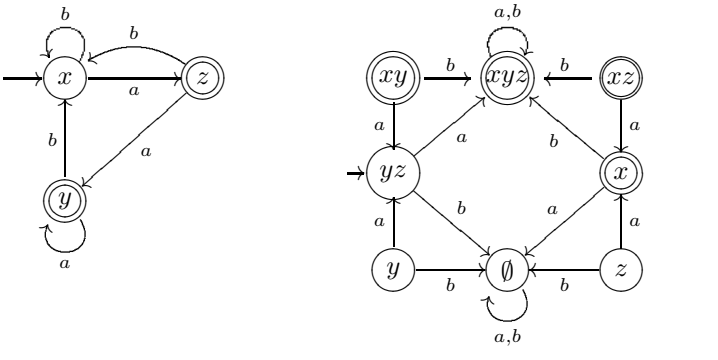
Theorem 2. *If the original automaton X is reachable, that is, if r is surjective, then the new automaton 2^X is observable, that is, O is injective. Furthermore, the language accepted by the initial state f of the new automaton 2^X is the reverse of the language accepted by the initial state i of the new automaton 2^X.*

Proof. As the operation $2^{(-)}$ transforms surjections into injections, reachability of X implies observability of 2^X. The second statement follows from the fact that we have

$$\begin{aligned} O(f) &= \{w \in A^* \mid 2^i(f_w) = 1\} \\ &= \{w^R \in A^* \mid i_w \in f\} \qquad \text{[by identity (7)]} \\ &= rev(\{w \in A^* \mid i_w \in f\}) \\ &= rev(o(i)) \end{aligned}$$

□

Example 3. We consider the following two automata. In the picture below, an arrow points to the initial state and a double circle indicates that a state is final:



(10)

The automaton on the left is reachable (but not observable, since y and z accept the same language $\{a, b\}^* a + 1$). Applying our construction above yields the

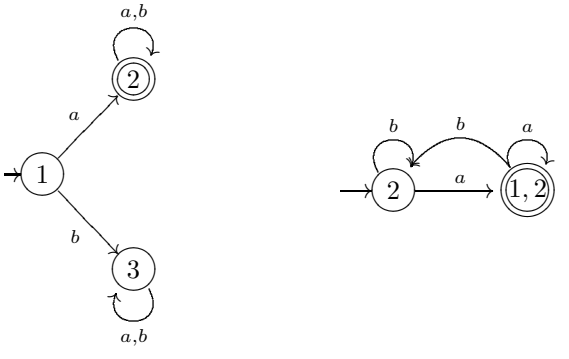
automaton on the right, which is observable (all the states accept different languages) but not reachable (e.g., the state $\{x, y\}$, denoted by xy, is not reachable from the initial state $\{y, z\}$). Furthermore, the language accepted by the state $\{y, z\}$ on the right: $a\{a, b\}^*$, is the reverse of the language accepted by the state x on the left, which is $\{a, b\}^*a$. □

4 Brzozowski's Algorithm

As an immediate consequence, we obtain the following version of Brzozowski's algorithm.

Corollary 4. *Applying the above construction to a deterministic and reachable automaton accepting a language L yields a minimal automaton accepting $rev(L)$. Taking of the latter automaton the reachable part, and applying the same procedure again yields a minimal automaton accepting L.*

Example 3 continued: We saw that applying our construction to the first automaton of Example 3 resulted in the second automaton given there. By taking the reachable part of the latter, we obtain the automaton depicted below on the left (where $1 = \{y, z\}$, $2 = \{x, y, z\}$ and $3 = \emptyset$):



(11)

The automaton on the right is obtained by applying our construction once more. It is the minimization of the automaton we started with. □

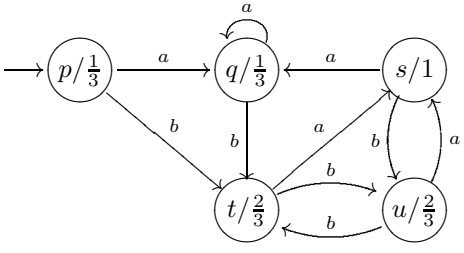
5 Moore Automata

Moore automata generalise deterministic automata by allowing outputs in an arbitrary set B, rather than just 2. Formally, a Moore automaton with inputs in A and outputs in B consists of a set of states X, an initial state $i\colon 1 \to X$, a transition function $t\colon X \to X^A$ and an output function $f\colon X \to B$. Moore automata accept functions in B^{A^*} (that is functions $\phi\colon A^* \to B$) instead of languages in 2^{A^*}.

Here is in a nutshell how our story above can be generalised to Moore automata. We can redraw diagram (2) by simply replacing 2 with B. We then

define reachability and observability as before. Next we adopt our procedure of reversing transitions by using (the contra-variant functor) $B^{(-)}$ instead of $2^{(-)}$: for all sets V, $B^V = \{\phi\colon V \to B\}$ and, for all functions $g\colon V \to W$, the function $B^g\colon B^W \to B^V$ maps each $\phi \in B^W$ to $B^g(\phi) = \phi \circ g$. Finally, all the results discussed above will also hold for Moore automata. The next example illustrates the minimization of a Moore automaton.

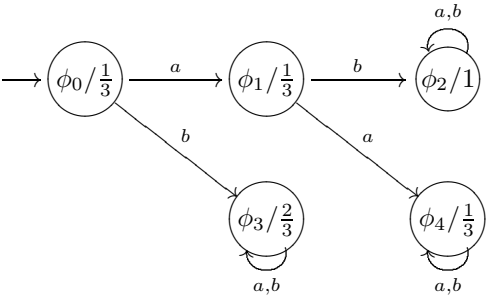
Example 5. We consider the following Moore automaton with inputs in $A = \{a, b\}$ and output in the set of real numbers $\mathbb{R}$. In the picture below, the output value r of a state s is indicated inside the circle by s/r:



(12)

The automaton accepts a function in $\mathbb{R}^{A^*}$ mapping every word w ending with ba to 1, every word ending with b to $\frac{2}{3}$ and every other word to $\frac{1}{3}$. Clearly the automaton is reachable from p. However it is not observable, since, for example, the states p and q accept the the same function.

Applying our construction above yields a Moore automaton with $\mathbb{R}^S$ as set of states, where $S = \{p, q, s, t, u\}$ is the set of states of the original automaton. The output value of a state $\phi\colon S \to \mathbb{R}$ is given by $\phi(p)$, where p is the initial state of the original automaton. Further, the output function of the original automaton becomes the new initial state, i.e., the function $\phi_0\colon S \to \mathbb{R}$ mapping p and q to $\frac{1}{3}$, t and u to $\frac{2}{3}$, and u to 1. By the above results, it follows that the automaton is observable. It is not reachable, as it contains infinitely many states, and there are only five states reachable from ϕ_0, as we can see in the picture below.

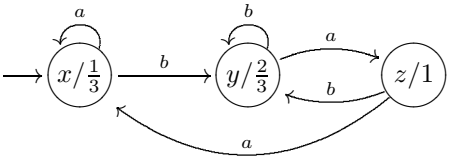


(13)

We do not spell out the full definition of the above states. As an example, the state ϕ_1 consists of the map assigning p, q and s to $\frac{1}{3}$, and t, u to 1. Note that the function in $\mathbb{R}^{A^*}$ accepted by the state ϕ_0 maps each words $w \in \{a, b\}^*$ to the same value where the reverse word w^R is mapped by the function accepted by

the original automaton in Figure 12. More formally, it maps words which begin with ab to 1, words which begin with b to $\frac{2}{3}$, and all other words to $\frac{1}{3}$.

If we repeat the same construction one more time, and take the reachable automaton from the initial state we obtain the minimal Moore automaton equivalent to the one in Figure 12:



(14)

□

6 Discussion

We have given a new description and a new correctness proof of Brzozowski's algorithm [6] for the minimization of deterministic automata. Next we have shown how to generalise the algorithm to deterministic Moore automata.

There are several other, sometimes more efficient minimization algorithms for finite automata [8]. But even though the complexity of Brzozowski's algorithm is double exponential in the worst case (essentially because it applies the contravariant powerset construction twice), the algorithm has been shown to behave rather well in practice, at least for non-deterministic automata [16].

Our proof of correctness of the algorithm is based on the duality between reachability and observability, which is due to Arbib and Manes [2,3]. Our formulation uses, albeit implicitly, a bit of (standard) universal algebra and coalgebra [12]. In particular we have given a simple proof that the resulting automaton is minimal, where minimality means that the automaton does not contain two different states with the same observable behaviour. Our method can be applied also to automata with infinitely many states.

The duality between reachability and observability has been studied also in other contexts. In [4], this duality is used to establish several analogies between concepts from observational (coalgebraic) and constructor-based (algebraic) specifications. In the present paper, we have highlighted how Brzozowski's algorithm integrates both concepts, for the specific case of deterministic and Moore automata.

Yet another approach, but somewhat similar in spirit to ours, can be found in [5], where a Stone-like duality between automata and their logical characterization is taken as a basis for Brzozowski's algorithm. The precise connection between that approach and the present paper, remains to be better understood. More generally, it is a challenge to try and generalise Brzozowski's algorithm to various other types of coalgebras.

Crucial for our approach was the combined use of both algebra and coalgebra. Notably, it was important to include in the definition of automaton an initial state, which is in essence an algebraic concept. In coalgebra, one typically models

automata without initial states. In that respect, so-called well-pointed coalgebras [1], which are coalgebras with a designated initial state, may have some relevance for the further generalization of Brzozowski's algorithm.

A somewhat different notion of nondeterministic Moore automata has recently been introduced in [7]. It basically consists of a nondeterministic automaton with output in a set that comes equipped with a commutative and associative operator. Interesting for our context is their variant of Brzozowski's algorithm for the construction of a minimal deterministic Moore automaton that is equivalent to a given nondeterministic one.

Brzozowski's minimization algorithm has also been combined with Brzozowski's method for deriving deterministic automata from regular expressions in [17]. It would be useful to investigate how such a combination can be generalized to, for example, Kozen's calculus of Kleene algebras with tests [9]. All that is required for our approach is a deterministic Moore automaton accepting the guarded language denoted by the reverse of the input expression.

Brzozowski's algorithm was originally formulated for *nondeterministic* finite automata [6]. Our present approach (as well as that of [5]) takes a *deterministic* automaton as a starting point. Recently [15], we have presented a generalisation of the subset construction for the determinisation of many different types of automata. Examples include Rabin's probabilistic automata [11] and weighted automata [14], to which our method applies in spite of the fact that the resulting deterministic (Moore) automaton is infinite. How to minimize nondeterministic automata directly, without having to introduce an extra determinisation step, is left as future work. Also we would like to combine the results of the present paper with those of [15] to generalise Brzozowski's algorithm to probabilistic and weighted automata.

Acknowledgements. Our interest in Brozowski's algorithm was raised by Prakash Panangaden, who asked himself (and several others) how Brzozowski's algorithm could be generalised to probabilistic automata. Jan Rutten and Alexandra Silva gratefully acknowledge several discussions with Prakash on this (and many other) subject(s). This work has been carried out under the Dutch NWO project *CoRE: Coinductive Calculi for Regular Expressions*. The work of Alexandra Silva was partially supported by Fundação para a Ciência e a Tecnologia, Portugal, under grant number `SFRH/BPD/71956/2010`. The work of Filippo Bonchi was partially supported by the PEPS-CNRS project CoGIP.

References

1. Adámek, J., Milius, S., Moss, L.S., Sousa, L.: Well-pointed Coalgebras (Unpublished note)
2. Arbib, M.A., Manes, E.G.: Adjoint machines, state-behaviour machines, and duality. Journal of Pure and Applied Algebra 6, 313–344 (1975)
3. Arbib, M.A., Manes, E.G.: Machines in a category. Journal of Pure and Applied Algebra 19, 9–20 (1980)

4. Bidoit, M., Hennicker, R., Kurz, A.: On the Duality between Observability and Reachability. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 72–87. Springer, Heidelberg (2001)
5. Bezhanishvili, N., Panangaden, P., Kupke, C.: Minimization via duality (Unpublished note)
6. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. Mathematical Theory of Automata 12(6), 529–561 (1962)
7. Castiglione, G., Restivo, A., Sciortino, M.: Nondeterministic Moore Automata and Brzozowski's Algorithm. In: Bouchou-Markhoff, B., Caron, P., Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2011. LNCS, vol. 6807, pp. 88–99. Springer, Heidelberg (2011)
8. Kozen, D.: Automata and Computability. Springer, Heidelberg (1997)
9. Kozen, D.: Kleene algebra with tests. ACM Trans. Program. Lang. Syst. 19, 427–443 (1997)
10. Hundt, C., Panangaden, P., Pineau, J., Precup, D., Dinculescu, M.: The duality of state and observations (Unpublished note)
11. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
12. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000); Fundamental Study
13. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009)
14. Schützenberger, M.P.: On the definition of a family of automata. Information and Control 4(2-3), 245–270 (1961)
15. Silva, A., Bonchi, F., Bonsangue, M.M., Rutten, J.J.M.M.: Generalizing the powerset construction, coalgebraically. In: Proc. of FSTTCS 2010. Leibniz International Proceedings in Informatics (LIPIcs) Series, vol. 8, pp. 272–283 (2010)
16. Watson, B.W.: Taxonomies and Toolkits of Regular Language Algorithms. Ph.D thesis, Eindhoven University of Technology, The Netherlands (1995)
17. Watson, B.W.: Directly Constructing Minimal DFAs: Combining Two Algorithms by Brzozowski. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 311–317. Springer, Heidelberg (2001)

Experimental Descriptive Complexity

Marco Carmosino[1,⋆], Neil Immerman[1,⋆], and Charles Jordan[2,⋆⋆]

[1] Computer Science Dept.
University of Massachusetts, Amherst
[2] Division of Computer Science
Hokkaido University
{mcarmosi,immerman}@cs.umass.edu, skip@ist.hokudai.ac.jp

Abstract. We describe our development and use of DescriptiveEnvironment (DE). This is a program to aid researchers in Finite Model Theory and students of logic to automatically generate examples, counter-examples of conjectures, reductions between problems, and visualizations of structures and queries.

DescriptiveEnvironment is available for free use under an ISC license at `http://www.cs.umass.edu/~immerman/de` . We encourage researchers and students at all levels to experiment with it. Please tell us of your insights, progress, suggestions, or extensions of DE.

Dedication

Dexter Kozen is a man of enormous energy. He plays soccer, rugby and ice hockey; he volunteers at the Cayuga Heights Fire Department; he raises three boys with Fran. In his spare time, Dexter proves theorems, teaches classes and writes books.

How has he done it all? One aspect is that when Dexter is engaged on a project, sleep is apparently unnecessary. Another is his plasticity: with an amazing ability to quickly master any relevant tool — whether it is a body of mathematics or a piece of software — that will add insight to what he is investigating, Dexter can shape shift easily. Between his flexibility, grace (watch out for him on the soccer field!) and his wide-ranging interests, Dexter has produced many elegant results. His results are deep and his impact has been wide.

We doubt that he is really turning sixty (In the words of Garnet Rogers, "What's Wrong With This Picture?" [Rog94][1]), but we wish him happy birthday all the same.

1 Introduction

When Neil Immerman was a graduate student at Cornell, he discovered Descriptive Complexity[2]. His dream was that rather than writing complex, error-prone

⋆ This research supported in part by NSF grants CCF 1115448; CCF 0830174.
⋆⋆ Supported by a Grant-in-Aid for JSPS Fellows under Grant No. 2100195209.

[1] Dexter has a rock band and it's easy to imagine them performing this song.

[2] Immerman originally called it "First Order Expressibility"; sometime later his advisor, Juris Hartmanis, suggested "Descriptive Complexity".

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 24–34, 2012.

programs, one could simply express the desired task in logic — leading to simple, correct and flexible code. However, it turns out that it is not particularly easy to write involved specifications in logic. Furthermore, just constructing examples and counter-examples of conjectures can be challenging. One tool to help with this would be a program that would automatically construct the models described by logical formulas. This is the genesis of DescriptiveEnvironment (DE).

For many years, DE was simply a glimmer in Immerman's eye. When Charles Jordan was an undergraduate at the University of Massachusetts in the early 2000s, he built a prototype of DE as a senior project. During his early years as a graduate student in computer science at Hokkaido University, Jordan refined DE. During a current visit to UMass Amherst, Jordan, along with graduate student Marco Carmosino, have extended DE.

In this paper we will introduce DE and present its basic functionality. In particular we will discuss **queries**: the formalism for expressing properties, transforming structures, and building reductions between problems. We begin by introducing some terminology from logic and Descriptive Complexity.

2 Descriptive Complexity: Mathematical Background

This background material is condensed from [Imm99]. The reader desiring more detail should consult that book as well as [EF99, Lib04].

This paper is not meant to be an introduction or survey on Descriptive Complexity. To satisfy the reader's curiosity we will provide the following very brief description of this subject. Beyond that, we point the reader to the following short survey: [Imm95], or to the above three books to find out about the subject in depth.

Descriptive Complexity began with Fagin's Theorem which says that a property, e.g., a graph property, is in NP iff it is expressible in second-order existential logic, i.e., NP = SO$\exists$ [Fag74]. Since that time, essentially all important complexity classes have been characterized via standard logical languages.

Two fundamental translations from computational complexity to logic are the following: (i) The parallel time needed to check whether an input structure has a property is equal to the depth needed to describe that property in a first-order inductive definition; and (ii) the amount of memory needed to check whether an input structure has a property is characterized by the number of distinct variables needed to describe that property in first-order logic.

In general, the computational complexity of checking whether an input has a given property can be exactly characterized as the richness of a logical language needed to express that property.

Thus the secrets of computation can be understood with logic. For this reason, it is of great use to have tools such as DE to help us manipulate and reason about the objects in question, i.e., structures and queries.

In the rest of this section we recall standard notation from mathematical logic. A **relational vocabulary**, $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ is a tuple of relation

symbols of given arity and constant symbols. For example, the following DE commands create the vocabularies "graph", consisting of one binary relation symbol, E, and two constant symbols, s, t; and "set" consisting of a single unary relations symbol, S.

```
graph is new vocabulary{E:2, s, t}.
set is new vocabulary{S:1}.
```

A **structure** with vocabulary τ is a tuple,

$$\mathcal{A} \quad = \quad \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$$

whose universe is the nonempty set $|\mathcal{A}|$. For each relation symbol R_i of arity a_i in τ, $\mathcal{A}$ has a relation $R_i^{\mathcal{A}}$ of arity a_i defined on $|\mathcal{A}|$, and for each constant symbol, c_j, $c_j^{\mathcal{A}}$ is an element of $|\mathcal{A}|$.

The following command creates line10, a structure of vocabulary graph that is a line graph on 10 vertices:

```
line10 is new structure{graph, 10, E:2 is x2=x1+1, s is 0, t is 9}.
```

In DE, the universe of a structure with size $n+1$ is $\{0, 1, \ldots, n\}$. The definition of E has two free variables, x1 and x2, and the above command defines the edge relation in line10 to be $\{(x_1, x_2) \mid 0 \leq x_1, x_2 \leq 9, x_2 = x_1 + 1\}$ (see Fig. 1). In DE, the definition of a relation symbol of arity a assumes that the free variables are $\{x_1, \ldots, x_a\}$.

The next instruction creates primes100, a structure of vocabulary set that is the set of all primes less than 100:

```
primes100 is new structure{set,100, S:1 is (1< x1 &
                  \A x, y.(x<=x1 & y<=x) : ((x*y=x1)->(x=1|y=1)))}.
```

The symbols `\A` and `\E` denote $\forall$ and $\exists$, and so S(x1) holds when x1 is prime.

Of course we can ask DE to print a given relation as follows, or to draw a given structure (Fig. 1).

```
primes100.S.
:{(2), (3), (5), (7), (11), (13), (17), (19), (23), (29), (31),
  (37), (41), (43), (47), (53), (59), (61), (67), (71), (73), (79),
  (83), (89), (97)}
```

2.1 Queries

A **query** is any mapping $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ from structures of one vocabulary to structures of another vocabulary, that is polynomially bounded. A **boolean query** is a map $I_b : \text{STRUC}[\sigma] \to \{0, 1\}$. In Descriptive Complexity, computations are queries and decision problems are boolean queries.

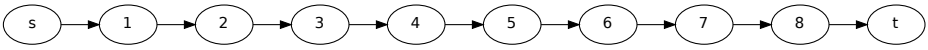


Fig. 1. The result of DE command: `draw(line10)`

Let σ and τ be any two vocabularies where $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. A **first-order query**, $I : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$,

$$I = \langle k, \varphi_0, \varphi_1, \ldots, \varphi_r, \psi_1, \ldots, \psi_s \rangle$$

is an $r + s + 2$-tuple consisting of a positive natural number k called the **dimension** of the query, plus formulas from the first-order language of σ, defining the universe of the image structure together with the relations and constants defined on the image structure.

For each structure $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, these formulas describe a structure $I(\mathcal{A}) \in \mathrm{STRUC}[\tau]$,

$$I(\mathcal{A}) \quad = \quad \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_s^{I(\mathcal{A})} \rangle \ .$$

The universe of $I(\mathcal{A})$ is a first-order definable subset of $|\mathcal{A}|^k$.

$$|I(\mathcal{A})| \quad = \quad \left\{ \langle b^1, \ldots, b^k \rangle \in |\mathcal{A}|^k \ \middle| \ \mathcal{A} \models \varphi_0(b^1, \ldots, b^k) \right\}$$

(Usually we will take $\varphi_0 \equiv \mathbf{true}$, thus letting $|I(\mathcal{A})|$ be the set of all k-tuples from $|\mathcal{A}|$.)

Each relation $R_i^{I(\mathcal{A})}$ is a first-order definable subset of $|I(\mathcal{A})|^{a_i}$,

$$R_i^{I(\mathcal{A})} \quad = \quad \left\{ (\langle b_1^1, \ldots, b_1^k \rangle, \ldots, \langle b_{a_i}^1, \ldots, b_{a_i}^k \rangle) \in |I(\mathcal{A})|^{a_i} \ \middle| \ \mathcal{A} \models \varphi_i(b_1^1, \ldots, b_{a_i}^k) \right\} .$$

Each constant symbol $c_j^{I(\mathcal{A})}$ is a first-order definable element of $|I(\mathcal{A})|$,

$$c_j^{I(\mathcal{A})} \quad = \quad \text{the unique } \langle b^1, \ldots, b^k \rangle \in |I(\mathcal{A})| \text{ such that } \mathcal{A} \models \psi_j(b^1, \ldots, b^k) \ .$$

For example, the following creates a binary first-order query I from graphs to sets. The universe formula for I is true, denoted in DE as `\t`:

```
I is new query{graph,string,2,\t,S:1 is E(x1,x2)}.
```

Applying I to line10 results in a set of 100 potential elements, populated by exactly the 9 edges of line10:

```
set100 is I(line10).
set100.S.
:{(1), (12), (23), (34), (45), (56), (67), (78), (89)}
```

Here is a slightly more interesting query:

```
R is new query{graph,graph,2,\t,E:2 is (x1=x3 & E(x2,x3))
                | (x1+1=x2 & x2=x3 &   x3=x4), s is x1=0 & x2=0,
                t is x1=max & x2=max}.
```

Here `max` denotes the maximum element of the universe. Let REACH be the set of graphs that have a path from s to t. Observe that for an undirected graph G, R(G) is in REACH iff G is connected. Thus R is a first-order reduction from connectivity of undirected graphs to REACH.

In general, if S and T are finite sets of structures of vocabulary σ and τ, respectively, and R is a first-order query from STRUC[σ] to STRUC[τ], then R is a **first-order reduction** from S to T iff for all finite structures $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$\mathcal{A} \in S \quad \Leftrightarrow \quad R(\mathcal{A}) \in T \; .$$

One of the reasons that complexity theory has been so successful in characterizing the complexity of problems is that naturally arising computational problems tend to be complete for important complexity classes such as NP, P, PSPACE, NSPACE[$\log n$], DSPACE[$\log n$], and a few others. This is true in spite of the fact that a well-known theorem of Ladner says that for any two of these classes that are distinct, there are intermediate problems, i.e., in but not complete for the larger class, but not in the smaller class [Lad75]. Contrast this with the fact that there are thousands of natural NP-complete problems, many dozens of P-complete problems, but only about four known natural problems that are in NP but not known to be in P nor NP complete.

A related phenomenon is that all those natural complete problems, originally shown complete via fairly powerful reductions, e.g., polynomial-time many-one reductions, tend to remain complete under first-order reductions. Remarkably, the extremely weak first-order projections (fop), quantifier-free reductions, and even quantifier-free projections (qfp) also usually suffice [Imm99, Val82].

For example, consider the first-order reduction R above. It is a qfp. It is quantifier-free because the definitions of the new relations and constants – E, s, t – are quantifier free. It is a projection because each bit of the output structure R(G) depends on at most one bit of the input structure G.

Relatively simple queries suffice to construct many of the objects of interest. Furthermore, it is possible to program via reductions, i.e., build a carefully constructed and optimized program for a complete problem, C, and then solve other problems by simply writing the reduction to C. An extremely successful example of this point of view occurs when C = SAT: We already have general automatic problem solving via SAT solvers.

Here is another example of the potential value of being able to reason about simple reductions. (See [Imm99] for details.) The problem REACH mentioned above is complete via qfps for the complexity class NSPACE[$\log n$] = NL. Let REACH_d be the subset of REACH containing graphs of out-degree at most one. (See Fig. 1 for a simple example.) REACH_d is complete via qfps for

$\mathrm{DSPACE}[\log n] = \mathrm{L}$. The following containments are well known and easy to prove:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}\,.$$

Everyone knows that is is open whether $\mathrm{P} = \mathrm{NP}$, but in fact it is open whether $\mathrm{L} = \mathrm{NP}$. Since three-colorability of graphs (3-COLOR) is complete for NP via logspace reductions (in fact, fops), it follows that 3-COLOR is reducible to REACH_d via qfps iff L= NP. In symbols,

$$\text{3-COLOR} \leq_{\mathrm{qfp}} \mathrm{REACH}_d \quad \Leftrightarrow \quad \mathrm{L} = \mathrm{NP}$$

Thus reasoning about first-order and quantifier-free reductions is valuable for proving upper and lower bounds on complexity.

3 Order-Independent P: A Motivating Example for DE

In 1982 Immerman and Vardi independently characterized polynomial time as the set of properties expressible in first-order logic plus the power to define new relations by induction: $\mathrm{P} = \mathrm{FO(LFP)}$ [Imm86, Var82].

But that was for ordered structures. When a graph or other logical structure is encoded in a computer, the vertices appear in some order. Furthermore, algorithms exploit this ordering by searching, for example, along the first edge leaving a particular vertex. To even the playing field, the languages we use to describe computational properties, starting with FO, include some arbitrary total ordering relation on the universe. (This is handled in DE by having the universe of all structures of size $n+1$ be $\{0, 1, \ldots, n\}$, where ordering, addition and multiplication on the integers is available.)

Graph properties are order-independent in the sense that they would not change if we changed the order in which the vertices are stored in your computer. However, since programs and formulas in FO(LFP) may use that arbitrary ordering, they may be computing a property of the ordered graph that depends on that ordering and so is not a graph property at all.

It is of great interest to capture **order-independent P** with a logical language. Whether this is possible remains open. If one simply removes ordering from FO(LFP), the resulting language is too weak. It cannot even count whether there are an even number of vertices in a graph. It was natural to ask whether $\mathrm{FO(LFP, COUNT)}$ – first-order logic with fixed point and counting quantifiers, but no ordering – captures order-independent P.

In 1981, Immerman and Eric Lander began investigating this question. It suffices to check it on graphs. They showed that this result holds for all trees and almost all graphs even when the formulas in question use only two variables [IL90].

If a graph is ordered, then each of its vertices has a unique name, i.e., the first vertex, the second vertex, etc. It is natural to consider graphs with unary relations representing colorings of the vertices. The *color-class size* of a graph is the number of vertices of the most popular color, e.g., if a graph had two

green vertices, one blue vertex, three red vertices and two yellow vertices then it would have color-class size 3. In some sense being ordered is the same as being of color-class size one.

In the paper [IL90], Immerman and Lander also showed that on graphs of color-class size at most 3, FO(LFP, COUNT) captures order-independent P, and three variables are necessary and sufficient.

Five years later, Jin-yi Cai (another Cornellian) and Immerman were trying to extend this result to larger color classes, in particular, trying to figure out whether this result held for color-class size 4 graphs. It doesn't; the CFI gadget discovered by Cai and Immerman and independently Fürer [CFI92], is a basis for the counterexample.

Fact 1. *There is a linear-time computable property of graphs of color-class size 4 that is not expressible in* FO(LFP, COUNT)*. To express this property for graphs on n vertices in* FO(LFP, COUNT) *requires $\Omega(n)$ distinct variables, while any fixed formula in* FO(LFP, COUNT) *only has a bounded number of variables.*

Although it remains open whether it is possible to capture order-independent P, there has been great progress on this problem. An operator giving the rank of a matrix makes the CFI property expressible, so FO(LFP, rank) is a strict extension of FO(LFP, COUNT) that is a new candidate for capturing order-independent P [DGH09]. Furthermore, in a break-through result, Grohe has shown that FO(LFP, COUNT) captures order-independent P for all classes of graphs having an excluded minor [Gro10].

4 Basic Functionality of DE

The CFI gadget took many blackboards, pieces of paper, and years to discover. With DE, we could have found it back then in minutes; today it would only take seconds. We have already seen a few DE commands. We now explain more of what DE can do.

As we have seen, DE lets us easily define vocabularies, structures, and queries. For example, let agraph be the vocabulary of graphs with two colors:

```
agraph is new vocabulary{A:1, E:2}.
```

In addition to defining structures explicitly as we did for line10 and set100 in §2, DE uses the tool Mace4[3] to generate a structure satisfying a given first-order sentence. For example, the following command calls Mace4 to create an undirected bipartite graph whose vertices all have degree at least 2:

```
bip2 is mace(agraph, \A x: \E y1,y2.y1!=y2: (E(x,y1) & E(x,y2)) &
                     \A x,y: (E(x,y)->(E(y,x) & (A(x)<->~A(y))))).
```

[3] Available at `http://www.cs.unm.edu/~mccune/prover9/`

The result is the smallest such graph: the complete bipartite graph $K_{2,2}$.

Built into DE are the boolean queries `minisat()` and `zchaff()` for calling SAT solvers MiniSat[4] and zChaff[5] to check the satisfiability of a propositional formula.

The vocabulary `sat` for propositional formulas is

```
sat is new vocabulary{P:2, N:2}.
```

Here $P(c, x_i)$ means that variable x_i occurs positively in clause c and $N(c, x_i)$ means that literal $\overline{x_i}$ occurs in c. The default is that a clause that has no literals is ignored.

DE makes it easy to use SAT solvers to solve a wide class of problems. For example, consider the following 3-ary reduction from 3COLOR to SAT:

```
thrcoltosat is new query{graph, sat, 3, x1<=3,
      P:2 is x1=3 & x2=0 & x3=x6 & x4=0 & x5<3,
      N:2 is x4=0 & E(x2,x3) & (x1=x5 & (x6=x2 | x6=x3)) & x1<3}.
```

Here, boolean variable $\langle 0, c, x\rangle$ means "vertex x is color c", clause $\langle 3, 0, x\rangle$ says that "vertex x is some color" and if $E(x, y)$ then clause $\langle a, x, y\rangle$ (for $0 \leq a \leq 2$) says that, "then x and y are not both color a".

We can then use the SAT solver to check 3-colorability. For example, if we have the DIMACS-format graph `myciel3.col`[6], we can use DE to check if it is 3-colorable:

```
myciel3 is load("myciel3.col").
mycielfmla is thrcolortosat(myciel3).
minisat(mycielfmla).
:\f
```

Here `mycielfmla` is a boolean formula that is satisfiable iff the graph myciel3 is 3-colorable. In this case, it is not satisfiable and thus we can conclude that myciel3 is not 3-colorable.

DE currently allows formulas from SO$\exists$(TC)[7], i.e., in addition to first-order logic, second-order existential logic, a transitive closure operator, and arithmetic are all available. Thus queries using TC and even second-order quantification may be written and evaluated. In the following two boolean queries, reach is defined using TC and isat is defined using second-order existential quantification. Not surprisingly, it is faster to use minisat and zchaff than to evaluate isat directly:

```
reach is new bquery{graph, TC[x,y:E(x,y)](s,t)}.

isat is new bquery{sat, \E S:1:\A z,x:\E y:(~P(z,x) & ~N(z,x)) |
                                (P(z,y) & S(y)) |(N(z,y) & ~S(y))}.
```

[4] `http://minisat.se`

[5] Available at `http://www.princeton.edu/~chaff/zchaff.html` . Due to licensing issues, users who wish to use zChaff must download it themselves.

[6] Available from `http://mat.gsia.cmu.edu/COLOR/instances.html`

[7] Except when using Mace4, when we are restricted to first-order.

5 A Motivating Example for ReductionFinder

Fact 2. [Imm88, Sze88] *For all* $s(n) \geq \log n$, NSPACE$[s(n)]$ = *co*-NSPACE$[s(n)]$.

When Immerman proved Fact 2, he was motivated by the fact that REACH is complete for NSPACE$[\log n]$ via quantifier-free projections (qfp) [Imm99]. Thus he knew that NSPACE$[\log n]$ (NL) is closed under complementation iff there is a qfp from $\overline{\text{REACH}}$ to REACH.

In fact, his original proof of Fact 2 was just that: a dimension 8 qfp from $\overline{\text{REACH}}$ to REACH. The construction, somewhat analogous to the much simpler reduction R from §2.1, proceeds by stepping through distances d, from 1 to n, and target vertices t, from 0 to $n-1$, computing the exact number n_d of vertices reachable from s in distance at most d, using the previously computed constant, n_{d-1}.

Immerman was struck at the time by how constraining – and thus useful – it was to have to build a qfp. It seemed either easy or impossible in the sense that at most steps of building the reduction there was only one possible thing to do.

This feeling led many years later to the building of ReductionFinder, a program that repeatedly uses a SAT solver to search for a small quantifier-free reduction between two problem specifications [CIE10]. At present writing, ReductionFinder is good at finding very small quantifier-free reductions – dimension 8 is way beyond the current capability. Still we feel that this approach has enormous potential. We are currently making the interface between DE and ReductionFinder.

6 Education

One of the differences between experts and students or novices is the experts' ability to visualize concepts relevant to their field. This ability aids in the construction and interpretation of diagrams, which allow communication between experts and discovery of new results. Students do not gain as much knowledge from examining domain-specific diagrams as experts do; they simply lack the context and skills to interpret diagrams.

In Descriptive Complexity, our diagrams are generally graphs depicting parts of logical structures before and after queries, or gadgets like the CFI construction. There are many high-quality open-source tools for drawing and labeling graphs. We have implemented a script that transforms the saved output of DE structures into files for GraphViz[8].

To make the process as easy as possible, we have added a "draw" command to DE, so that users can make changes to defined structures and then immediately observe their effect. This process mimics the visualization skills of the expert, and will allow students to gain a deeper understanding of how structures are defined by logical formulas.

[8] `http://www.graphviz.org/`

In the future, we will extend "draw" to operate on queries, showing a before-and-after view of the logical structure, and a typeset presentation of the relevant logical formulas in one image. This will not only help students understand example queries through modifying and plotting them, it will help researchers examine and verify the results of more complex queries.

Ehrenfeucht-Fraïssé games and their generalizations are a vital tool for Descriptive Complexity. It would be easier to teach students to play these games if we had software to play them automatically. We will build an EF-game engine for DE, because it is convenient to do so: all of the relevant datatypes are already available. Playing against an automated Samson or Delilah, each step of the game will be drawn to the screen. We expect that students will develop an intuition for EF-games and their generalizations faster and easier when a computer implementation of the game with graphics is readily available.

In the longer term, DE can be used as the backend for a web-based system. This would be the most convenient and easy format for students, and would allow the use of a rich web UI for playing EF games and playing with example queries and structures.

7 Conclusions

We have explained some of the motivations for building DE, together with some of its functionality. We encourage anyone to use it. Please send us your examples, questions, suggestions, improvements, and extensions.

In the near future we anticipate adding more features and visualization tools. In particular, we can hardly wait until DE is able to play Ehrenfeucht-Fraïssé games. It will also aid in visualizing a query by drawing the input and output structures, and letting the user choose points in the input or output and having the display highlight where they lead to or come from. We will add sliders to help students and researchers visualize the computations of transitive closures and fixed points.

There are many great programs and systems related to logic that are being developed and used that we have omitted in this introduction to DE. Our plan is to have an easy-to-use, extensible testbed. If there are great tools available, such as SAT solvers, Mace, ReductionFinder, that we can call to make DE more useful, then we are delighted to do so. The intent of DE is to make the research and teaching of logical methods easier and more fun.

References

[CFI92] Cai, J., Fürer, M., Immerman, N.: An Optimal Lower Bound on the Number of Variables for Graph Identification. Combinatorica 12(4), 389–410 (1992)

[CIE10] Crouch, M., Immerman, N., Moss, J.E.B.: Finding Reductions Automatically. In: Blass, A., Dershowitz, N., Reisig, W. (eds.) Fields of Logic and Computation. LNCS, vol. 6300, pp. 181–200. Springer, Heidelberg (2010)

[EF99] Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)

[DGH09] Dawar, A., Grohe, M., Holm, B., Laubner, B.: Logics with Rank Operators. In: IEEE Symp. Logic In Comput. Sci., pp. 113–122 (2009)

[Fag74] Fagin, R.: Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In: Karp, R. (ed.) Complexity of Computation. SIAM-AMS Proc., vol. 7, pp. 43–73 (1974)

[Gro10] Grohe, M.: Fixed-Point Definability and Polynomial Time on Graphs with Excluded Minors. In: IEEE Symp. Logic In Comput. Sci., pp. 179–188 (2010)

[Imm99] Immerman, N.: Descriptive Complexity. Springer Graduate Texts in Computer Science, New York (1999)

[Imm95] Immerman, N.: Descriptive Complexity: A Logician's Approach to Computation. Notices of the American Mathematical Society 42(10), 1127–1133 (1995)

[Imm88] Immerman, N.: Nondeterministic Space is Closed Under Complementation. SIAM J. Comput. 17(5), 935–938 (1988)

[Imm87] Immerman, N.: Languages That Capture Complexity Classes. SIAM J. Comput. 16(4), 760–778 (1987)

[Imm86] Immerman, N.: Relational Queries Computable in Polynomial Time. Information and Control 68, 86–104 (1986); A preliminary version of this paper appeared in ACM Symp. Theory of Comput., pp. 147–152 (1982)

[IL90] Immerman, N., Lander, E.S.: Describing Graphs: A First-Order Approach to Graph Canonization. In: Selman, A. (ed.) Complexity Theory Retrospective, pp. 59–81. Springer, Heidelberg (1990)

[Lad75] Ladner, R.: On the structure of polynomial time reducibility. J. Assoc. Comput. Mach. 22(1), 155–171 (1975)

[Lib04] Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)

[Rog94] Rogers, G.: What's Wrong With This Picture?, `http://www.garnetrogers.com/lyrics/What's%20Wrong%20With%20This%20Picture.txt`

[Sze88] Szelepcsényi, R.: The Method of Forced Enumeration for Nondeterministic Automata. Acta Informatica 26, 279–284 (1988)

[Var82] Vardi, M.: Complexity of Relational Query Languages. In: ACM Symp. Theory of Comput., pp. 137–146 (1982)

[Val82] Valiant, L.: Reducibility By Algebraic Projections. L'Enseignement mathématique T. XXVIII(3-4), 253–268 (1982)

Meditations on Quantified Constraint Satisfaction

Hubie Chen

Departament de Tecnologies de la Informació i les Comunicacions
Universitat Pompeu Fabra
Barcelona, Spain
hubie.chen@upf.edu

Abstract. The quantified constraint satisfaction problem (QCSP) is the problem of deciding, given a structure and a first-order prenex sentence whose quantifier-free part is the conjunction of atoms, whether or not the sentence holds on the structure. One obtains a family of problems by defining, for each structure $\mathbf{B}$, the problem $\mathsf{QCSP}(\mathbf{B})$ to be the QCSP where the structure is fixed to be $\mathbf{B}$. In this article, we offer a viewpoint on the research program of understanding the complexity of the problems $\mathsf{QCSP}(\mathbf{B})$ on finite structures. In particular, we propose and discuss a group of conjectures; throughout, we attempt to place the conjectures in relation to existing results and to emphasize open issues and potential research directions.

This article is dedicated to Dexter Kozen on the occasion of his 60th birthday. Congratulations and thanks, Dexter, for all that you have given us.

1 Introduction

The constraint satisfaction problem (CSP) is a general computational problem that involves deciding, given a set of constraints on a set of variables, whether or not there is an assignment to the variables satisfying all of the constraints. Problems from many areas of computer science can be formulated within this general framework, including graph coloring problems, boolean satisfiability problems, and problems from temporal and spatial reasoning. The CSP can be formalized logically as the problem of deciding, given a prenex $\{\exists, \wedge\}$-sentence Φ and a structure $\mathbf{B}$, whether or not Φ holds on $\mathbf{B}$. By a $\{\exists, \wedge\}$-sentence, we mean a first-order sentence built from atoms, the connective $\wedge$, and existential quantification. Deciding such a sentence in prenex form amounts to deciding if there is an assignment to the variables (each of which is existentially quantified) that satisfies all of the atoms, which can be thought of as constraints on the variables. The quantified constraint satisfaction problem (QCSP) is the natural generalization of the CSP in which universal quantification is also permitted: it is the problem of deciding, given a prenex $\{\forall, \exists, \wedge\}$-sentence Φ and a structure $\mathbf{B}$, whether or not Φ holds on $\mathbf{B}$. The addition of the universal quantifier permits the expression

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 35–49, 2012.

of a wider range of problems, including problems from quantified propositional logic and game theory. The greater expressiveness of the QCSP comes at the cost of higher computational complexity: if one considers finite structures, the CSP is in general NP-complete, while the QCSP is in general PSPACE-complete.

For each of these two problems, one can obtain restricted cases of interest by considering the problem relative to a fixed structure. Formally, for each structure $\mathbf{B}$, define $\mathsf{CSP}(\mathbf{B})$ (respectively, $\mathsf{QCSP}(\mathbf{B})$) to be the problem of deciding, given a $\{\exists, \wedge\}$-sentence Φ (respectively, $\{\forall, \exists, \wedge\}$-sentence Φ), whether or not Φ holds on $\mathbf{B}$. As examples, consider 2-SAT and Horn-SAT, two well-known polynomial-time tractable cases of the boolean satisfiability problem. One can capture these two problems as problems of the form $\mathsf{CSP}(\mathbf{B})$, by defining appropriate structures $\mathbf{B}$; correspondingly, the quantified generalizations of these problems, Quantified 2-SAT and Quantified Horn-SAT, can each be formulated as a problem of the form $\mathsf{QCSP}(\mathbf{B})$.

A research direction with roots in seminal articles by Schaefer [31] and Feder and Vardi [19] is to understand the complexity behavior of the problem family $\mathsf{CSP}(\mathbf{B})$ on finite structures $\mathbf{B}$. Schaefer [31] classified the complexity of $\mathsf{CSP}(\mathbf{B})$ on two-element structures: he described the two-element structures $\mathbf{B}$ such that $\mathsf{CSP}(\mathbf{B})$ is in P, showing that for all other two-element structures $\mathbf{B}$, the problem $\mathsf{CSP}(\mathbf{B})$ is NP-complete. Feder and Vardi [19] studied the problems $\mathsf{CSP}(\mathbf{B})$ on general finite structures, and famously conjectured that this problem family admits a dichotomy in that each problem therein is either in P or is NP-complete. In the 90s, Jeavons and co-authors pioneered an algebraic approach to studying the problems $\mathsf{CSP}(\mathbf{B})$, which involves associating an algebra to each finite structure $\mathbf{B}$, and then using properties of the algebra of $\mathbf{B}$ to gain insight into and derive results on the complexity of $\mathsf{CSP}(\mathbf{B})$; see for example [24, 23, 22, 8]. Although this article's focus is on finite structures, it can be remarked that this algebraic approach has been developed by Bodirsky and co-authors for ω-categorical structures [5, 4, 3].

The algebraic approach to studying the family $\mathsf{CSP}(\mathbf{B})$ has been the subject of focused research in the 00s which continues to the present (see for example [9, 26, 27, 1, 21] and the references therein). As the set of tools for understanding this family has developed, researchers have also studied many variants of the CSP with the aim of giving full complexity classifications on all finite structures $\mathbf{B}$. One example variant is the QCSP; as another, one can name the *counting CSP*, wherein one wants to count the number of satisfying assignments, in place of deciding if one exists [10]. (See [25, 18, 7] for further examples.) Complexity classification results typically give broad sufficient conditions for tractability, intractability, or (more generally) completeness for a complexity class; they can often be used as the basis for analyzing the complexity of further problems.

In this article, we offer a viewpoint on the research direction of understanding the complexity of the problem family $\mathsf{QCSP}(\mathbf{B})$ over finite structures. In particular, we propose and discuss a group of conjectures concerning the complexity of the problems $\mathsf{QCSP}(\mathbf{B})$. Throughout our presentation, we attempt to place the conjectures in relation to existing results, and to emphasize open issues and

potential research directions. One of the leading protagonists of these conjectures is a property of algebras called the *polynomially generated powers (PGP) property*: essentially, an algebra $\mathbb{A}$ has this property when its powers $\mathbb{A}, \mathbb{A}^2, \mathbb{A}^3, \ldots$ have generating sets of polynomial size. This property is a close relative of the *few subpowers property*, another property of algebras that concerns a growth rate in the powers of an algebra and which has given insight into the CSP [2, 21]; see [17] for a discussion. We point out that in the case of the CSP, a precise conjecture predicting the boundary of the P/NP-complete dichotomy was posed in the early 00s, in the conference version of [8]; inside P, a picture of the situation has emerged with demonstrations of general hardness criteria for sub-P complexity classes [27].

2 Preliminaries

We study relational first-order logic. A *signature* is a set of relation symbols, each of which has an associated arity. A *relational structure* $\mathbf{B}$ over a signature σ consists of a set B called the *universe* and a relation $R^{\mathbf{B}} \subseteq B^k$ for each relation symbol $R \in \sigma$; here, k denotes the arity of R. We will frequently use the symbol $\mathbf{B}$ to denote a relational structure with universe B. In this article, we focus on finite relational structures, by which is meant structures having finite universes. Let $\mathbf{B}$ be a structure on signature σ. We use $\mathbf{B}^*$ to denote the expansion of $\mathbf{B}$ containing all constant relations; precisely, $\mathbf{B}^*$ is a structure over signature $\sigma^* \supseteq \sigma$ with universe B such that (1) for each $b \in B$, there exists a relation symbol $R_b \in \sigma^*$ of arity 1 where $R_b^{\mathbf{B}^*} = \{b\}$, and (2) each relation symbol in $\sigma^* \setminus \sigma$ is of the form R_b where $R_b^{\mathbf{B}^*} = \{b\}$ and such that for no other relation symbol $S \in \sigma^*$ does it hold that $S^{\mathbf{B}^*} = \{b\}$. Note that we have $(\mathbf{B}^*)^* = \mathbf{B}^*$.

Let σ be a signature. We define a *quantified constraint formula* over σ to be a first-order formula of the form $Q_1 v_1 \ldots Q_n v_n \phi$ where each $Q_i \in \{\forall, \exists\}$ is a quantifier, each v_i is a variable, and ϕ is the conjunction of σ-predicate applications. By a σ-predicate application, we mean a formula $R(w_1, \ldots, w_k)$ where $R \in \sigma$, the w_i are variables, and k is the arity of R. We define a *constraint formula* to be a quantified conjunctive formula that does not make use of universal quantification, that is, where all quantifiers are existential. We will use the term *(quantified) constraint sentence* to refer to a (quantified) constraint formula having no free variables.

The CSP (QCSP) is the problem of deciding, given a (quantified) constraint sentence Φ and a structure $\mathbf{B}$, whether or not $\mathbf{B} \models \Phi$. Each structure gives rise to a restricted version of each of these problems, defined as follows. For a structure $\mathbf{B}$, the problem $\mathsf{CSP}(\mathbf{B})$ is the problem of deciding, given a constraint sentence Φ, whether or not $\mathbf{B} \models \Phi$; similarly, the problem $\mathsf{QCSP}(\mathbf{B})$ is the problem of deciding, given a quantified constraint sentence Φ, whether or not $\mathbf{B} \models \Phi$. We will be interested in studying the QCSP where, in addition to fixing the structure, the number of quantifier alternations is bounded. We define a representative sequence of problems as follows. For each $k \geq 1$, we define Π_{2k}-$\mathsf{QCSP}(\mathbf{B})$ to be the restriction of the problem $\mathsf{QCSP}(\mathbf{B})$ to quantifier prefixes that are Π_{2k}.

Note that the only notion of complexity-theoretic reduction that we will make use of in this paper is many-one polynomial-time reduction.

Let B be a nonempty set, let $f : B^n \to B$ be an n-ary operation on B, and let $R \subseteq B^k$ be a k-ary relation on B. We say that f is a *polymorphism* of R (and that f *preserves* R) if for every length n sequence of tuples $t_1, \ldots, t_n \in R$, denoting the tuple t_i by $(t_{i,1}, \ldots, t_{i,k})$, it holds that the tuple

$$f(t_1, \ldots, t_n) = (f(t_{1,1}, \ldots, t_{n,1}), \ldots, f(t_{1,k}, \ldots, t_{n,k}))$$

is in R. We extend this terminology to relational structures, and say that an operation f is a polymorphism of a relational structure $\mathbf{B}$ if f is a polymorphism of every relation of $\mathbf{B}$.

For our purposes, an *algebra* $\mathbb{A}$ is a pair (A, F) consisting of a non-empty set A called the *universe* and a set F of finitary operations on A. Let $\mathbb{A} = (A, F)$ be an algebra. A *subalgebra* of $\mathbb{A}$ is an algebra of the form $(B, F|_B)$, where $B \subseteq A$ and B is preserved by all operations in F. By $F|_B$, we mean the set $\{f|_B \mid f \in F\}$, where $f|_B$ denotes the restriction of the operation f to the set B. The subalgebra (of $\mathbb{A}$) *generated* by a subset $X \subseteq A$ is defined to be the intersection of all $\mathbb{A}$-subalgebras containing X. We say that a subset $X \subseteq A$ generates an algebra $\mathbb{A} = (A, F)$ if $\mathbb{A}$ itself is the subalgebra of $\mathbb{A}$ generated by X. We will frequently use the symbol $\mathbb{A}$ to denote an algebra (A, F).

Our focus in this paper is on finite structures of the form $\mathbf{B}^*$, that is, finite structures having a relation for each constant. We thus define $\mathbb{A}_\mathbf{B}$, the algebra for structure $\mathbf{B}$, to be the algebra $(B, \mathsf{IPol}(\mathbf{B}))$ where $\mathsf{IPol}(\mathbf{B})$ denotes the set of all idempotent polymorphisms of $\mathbf{B}$. Recall that an operation $f : B^n \to B$ is idempotent if for all $b \in B$, it holds that $f(b, \ldots, b) = b$. Note that for any structure $\mathbf{B}$, it holds that $\mathsf{IPol}(\mathbf{B}) = \mathsf{IPol}(\mathbf{B}^*)$ and hence $\mathbb{A}_\mathbf{B} = \mathbb{A}_{\mathbf{B}^*}$. For a finite structure $\mathbf{B}^*$, the set of idempotent polymorphisms $\mathsf{IPol}(\mathbf{B}^*)$ can be connected to the complexity of $\mathsf{QCSP}(\mathbf{B}^*)$; see [6].

In the case of the CSP, it is known that for each finite structure $\mathbf{A}$, there exists a finite structure $\mathbf{B}$ such that the problems $\mathsf{CSP}(\mathbf{A})$, $\mathsf{CSP}(\mathbf{B}^*)$ are polynomial-time interreducible [8]. In the context of CSP complexity classification, this result justifies focusing on structures having constants, that is, structures of the form $\mathbf{B}^*$. In contrast, for the QCSP, no such passage to structures having constants is known. From this author's perspective, any results contributing to an understanding of whether or not such a passage exists would be highly appreciated. Perhaps the discussion in the last section of this article will provide some clues!

3 The Polynomially Generated Powers Property

For an algebra $\mathbb{A}$, let $d(\mathbb{A})$ denote the smallest size of a generating set for $\mathbb{A}$. The sequence $(d(\mathbb{A}), d(\mathbb{A}^2), d(\mathbb{A}^3), \ldots)$ is called the *d-sequence* of an algebra $\mathbb{A}$. Study of the d-sequence goes back to work of Wiegold and co-authors [32–35, 29, 36], which focused on groups and semigroups. Recent years have seen a revival of interest in the d-sequences of algebras; see for example [30, 20]. Previous work by the present author [17] connected the complexity of $\mathsf{QCSP}(\mathbf{B})$ with polynomially

bounded growth of the d-sequence of $\mathbb{A}_{\mathbf{B}}$; this mode of growth is formalized as follows.

Definition 1. *An algebra* $\mathbb{A}$ *has the* polynomially generated powers (PGP) property *if there exists a polynomial* $p(n)$ *(on the natural numbers) such that for all* $n \geq 1$*, there exists a subset* $X_n \subseteq A^n$ *of size* $|X_n| \leq p(n)$ *that generates the algebra* $\mathbb{A}^n$*.*

In particular, one can derive implications for the QCSP when the polymorphism algebra has the PGP property in an effective sense; effectiveness of this property is formalized as follows.

Definition 2. *An algebra* $\mathbb{A}$ *has the* effective PGP property *if there exists an algorithm that, given a natural number* $n \geq 1$*, outputs in polynomial time (in* n*) a subset* $X_n \subseteq A^n$ *that generates the algebra* $\mathbb{A}^n$*.*

Remark 1. We will discuss the effective PGP property only on finite algebras. We require that the algorithm in the definition outputs each set X_n in the form of an explicit listing of n-tuples. Since in polynomial time it is only possible to output polynomially many tuples, the effective PGP property implies the PGP property.

Example 1. Consider an algebra $\mathbb{A}$ with universe B having a binary operation $f : B^2 \to B$ with respect to which there is an identity element, that is, such that there exists $e \in B$ such that for all $b \in B$, it holds that $f(b, e) = f(e, b) = b$. We will argue that such an algebra $\mathbb{A}$ has the PGP. Monoids and groups are examples of algebras having this property.

For each $n \geq 1$, for each $i \in \{1, \ldots, n\}$, and for each $b \in B$, define $t[n, i, b]$ to be the tuple in B^n whose ith entry is equal to b, and which has all other entries equal to e. For each $n \geq 1$, define T_n to be the set

$$\{t[n, i, b] \mid i \in \{1, \ldots, n\}, b \in B\}.$$

We claim that, with respect to the algebra $\mathbb{A}$, the set T_n generates B^n. Let $(b_1, \ldots, b_n)$ be an arbitrary element of B^n. By f-multiplying together the tuples $t[n, 1, b_1], t[n, 2, b_2], \ldots, t[n, n, b_n]$ in any order, one arrives at $(b_1, \ldots, b_n)$, establishing the claim. We have $|T_n| \leq n|B|$, that is, the size of the sets T_n exhibits linear growth, and hence the algebra $\mathbb{A}$ has the PGP with respect to the polynomial $p(n) = n|B|$. Moreover, one can readily devise a polynomial-time algorithm that, given n, outputs T_n, and thus the algebra $\mathbb{A}$ has the effective PGP.

Recall that a *semilattice operation* is a binary operation that is associative, commutative, and idempotent. On the Boolean universe $\{0, 1\}$, there are two semilattice operations, the Boolean AND ($\wedge$), which has identity element 1, and the Boolean OR ($\vee$), which has identity element 0. Consider the 3-element algebra $\mathbb{A} = (\{a, b, c\}, \{f\})$ where $f : \{a, b, c\}^2 \to \{a, b, c\}$ is defined by the following rule: $f(x, y) = x$ if $x = y$, and $f(x, y) = c$ otherwise. The operation f is a semilattice operation. It can be verified that, for any generating set T_n of $\mathbb{A}^n$, it holds that $T_n \supseteq \{a, b\}^n$; this follows from the fact that if f is applied to two n-tuples s, s' and

the result is a tuple $t \in \{a, b\}^n$, then $s = s' = t$. Consequently, any such generating set T_n must have $|T_n| \geq 2^n$, and this 3-element algebra $\mathbb{A}$ does not have the PGP. More generally, it can be shown that if a semilattice operation $f : B^2 \to B$ does not have an identity element, then for an appropriate choice of distinct elements $a, b \in B$ one again has that any generating set of B^n must contain $\{a, b\}^n$, and thus that the algebra $(B, \{f\})$ does not have the PGP. □

The following theorem directly connects the QCSP to the effective PGP property. Essentially, it states that when the algebra $\mathbb{A}_{\mathbf{B}}$ has this property, the bounded-alternation QCSP on $\mathbf{B}$ can be reduced to $\mathsf{CSP}(\mathbf{B}^*)$ and is hence in NP.

Theorem 1. *[17] Let* $\mathbf{B}$ *be a finite relational structure on a finite signature. If the algebra* $\mathbb{A}_{\mathbf{B}}$ *has the effective PGP property, then for all* $k \geq 1$*, the problem* Π_{2k}-$\mathsf{QCSP}(\mathbf{B})$ *reduces to* $\mathsf{CSP}(\mathbf{B}^*)$ *and is in NP.*

We give an explanation of how this theorem is proved.

Proof idea of Theorem 1. It suffices to prove the result under the assumption that $\mathbf{B} = \mathbf{B}^*$, which is what we do. The proof is an induction argument showing that, for all $k \geq 1$, there exists a polynomial-time algorithm that converts a Π_{2k} quantified constraint formula Φ to a constraint formula Φ' that is equivalent in the natural sense: a B-assignment f to the free variables satisfies Φ over $\mathbf{B}$ if and only if it satisfies Φ' over $\mathbf{B}$.

For $k = 1$, this is done as follows. Let Y denote the universally quantified variables in Φ. A generating set $\{t_1, \ldots, t_m\}$ for $\mathbb{A}_{\mathbf{B}}^{|Y|}$ is computed in polynomial time using the algorithm giving the effective PGP property; each element t_i in the set can (and will!) be naturally viewed as a mapping from Y to B. For each i from 1 to m, a formula Φ_i is created from Φ by changing all universal quantifiers to existential quantifiers, and then conjoining $\bigwedge_{y \in Y} R_{t_i(y)}(y)$ to the quantifier-free part. In the formula Φ_i, the universally quantified variables (of Φ) are forced to be set according to the generator t_i. The desired formula Φ' is (the prenexing of) the formula $\bigwedge_{i=1}^{m} \Phi_i$. The point is that to determine the truth of Φ (relative to $\mathbf{B}$ and an assignment to the free variables), in lieu of considering *all* possible assignments to the universally quantified variables Y, one can consider just the assignments t_i from the generating set. (We note that verifying this makes use of the idempotence of the operations of $\mathbb{A}_{\mathbf{B}}$.) In essence, having a polynomially-sized generating set allows us to short-cut the consideration of the exponentially many assignments to the universally quantified variables.

The induction is argued as follows. Suppose that one has the result for k; we want to show the result for $k+1$. Given a $\Pi_{2(k+1)}$ quantified constraint formula Φ, by removing the outermost two quantifier blocks, we obtain a Π_{2k} formula Φ_2. We use the algorithm for Π_{2k} formulas, which exists by induction, to obtain an equivalent constraint formula. Taking this constraint formula and putting back the two quantifier blocks that were removed, we obtain a Π_2 formula, which can then be handled using the algorithm presented for the case $k = 1$. The overall algorithm in this case, modulo some simple syntactic manipulations, is the composition of two polynomial-time algorithms, and is hence itself polynomial-time. □

The argument just given certainly allows one (under the assumption of the effective PGP property) to translate any instance Φ of $\mathsf{QCSP}(\mathbf{B})$ to a truth-equivalent instance of $\mathsf{CSP}(\mathbf{B}^*)$: simply determine the lowest k such that Φ is Π_{2k}, and then apply the algorithm given for Π_{2k} formulas. However, it is worth noting that the action of this translation on *all* instances of $\mathsf{QCSP}(\mathbf{B})$ is *not* in polynomial time. We give a brief explanation as to why. Consider an algebra $\mathbb{A}$ that has no generating set of size 1, that is, where every generating set has size greater than or equal to 2. Let us examine how the translation acts on a Π_{2n} formula of the form $\forall y_1 \exists x_1 \ldots \forall y_n \exists x_n \phi$. The translation works from the inside out; first, the subformula $\forall y_n \exists x_n \phi$ is converted to a constraint formula ϕ_1. The resulting constraint formula arises as a conjunction (of copies of ϕ) over a generating set for $\mathbb{A}$, and so by assumption has size at least 2. Then, the translation converts the formula $\forall y_{n-1} \exists x_{n-1} \phi_1$ to a constraint formula ϕ_2; again, the result is a conjunction of copies of ϕ_1 over a generating set for $\mathbb{A}$, and so has size at least 4. Continuing to argue in this way, one obtains that the size of each intermediate formula ϕ_i has size at least 2^i, and the final constraint formula ϕ_n has size at least 2^n.

Our first conjecture is that each finite algebra has either the PGP property or has a d-sequence with exponential growth; the latter property is formalized as follows.

Definition 3. *An algebra* $\mathbb{A}$ *has the* exponentially generated powers (EGP) property *if there exists a real number* $b > 1$ *such that the mapping (on the positive natural numbers) that takes* n *to* $d(\mathbb{A}^n)$ *is* $\Omega(b^n)$.

Note that a dichotomy between the PGP property and the EGP property cannot be taken for granted, as there are growth rates (such as $n^{\log n}$) that are neither polynomial nor exponential, according to the definitions here.

Conjecture 1. Each finite algebra $\mathbb{A}$ either has the PGP property or has the EGP property.

Support for this conjecture can be found, for example, in the article of Wiegold [36] on semigroups, and in previous work relating the PGP property to the QCSP, where a class of 3-element algebras was studied [17].

A parenthesis. We will, later in this paper, conjecture that absence of the PGP property leads to coNP-hardness of the QCSP (see Conjecture 3 and Remark 4). Intuitively speaking, existing coNP-hardness results in this vein (which sometimes apply to structures having a tractable CSP) use a block of universally quantified variables to induce the consideration of an exponentially large search space, in such a way that the quantified constraint sentence is true if and only if there is no object of desirable type in the search space. Conjecture 1 is algebraic: we are not aware of any direct implications that it would have for QCSP complexity. But, could belief in the computational Conjecture 3 support belief in the algebraic Conjecture 1?

4 Bounded Alternation

In this section, we present and discuss three conjectures in the setting of bounded alternation. The first we view as quite innocuous: it states that, on finite algebras, the PGP property is always effective. The truth of this conjecture would allow us to directly connect the PGP property to the bounded-alternation QCSP.

Conjecture 2. For every finite algebra $\mathbb{A}$, if $\mathbb{A}$ has the PGP property, then $\mathbb{A}$ has the effective PGP property.

Remark 2. From a proof of Conjecture 2, one would be able to simplify the statement of Theorem 1: one could remove the effectiveness assumption on the PGP property.

Remark 3. Previous work on the QCSP [17] identified a property on algebras called *switchability* which implies the PGP; in fact, a look at the generating sets given by switchability allows one to readily verify that switchability implies the effective PGP. Within a particular family of 3-element algebras, it was shown that the PGP implies switchability, providing evidence for Conjecture 2. We believe that it could be of interest to investigate whether or not all algebras having the PGP property are switchable.

So, Conjecture 2 predicts that the presence of the PGP property places the bounded-alternation QCSP in NP. Our next conjecture predicts that the PGP property is, in fact, the only explanation for this QCSP being in NP, on structures having the form $\mathbf{B}^*$.

Conjecture 3. Let $\mathbf{B}$ be a finite relational structure on a finite signature. If the algebra $\mathbb{A}_{\mathbf{B}}$ does not have the PGP property, then there exists $k \geq 1$ such that Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ is coNP-hard.

Remark 4. Examples of coNP-hardness results for the QCSP which provide evidence for Conjecture 3 can be found in the articles [11–13].

The two conjectures given in this section (thus far) predict when the bounded-alternation QCSP on structures $\mathbf{B}^*$ will be in NP. Indeed, these conjectures predict that if the sequence Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ is in NP, then the algebra $\mathbb{A}_{\mathbf{B}} = \mathbb{A}_{\mathbf{B}^*}$ has the PGP and each problem in this sequence reduces to (and is hence equivalent to) $\mathsf{CSP}(\mathbf{B}^*)$, as in Theorem 1. Outside of NP, one can observe two different modes of complexity behavior (for the QCSP under bounded alternation). One is coNP-completeness. An example of this behavior is given by certain structures preserved by semilattice operations without an identity element, for which one has, for each $k \geq 1$, coNP-completeness of Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$: coNP-hardness comes from [11], and containment in coNP follows from results in [15]. Another behavior is that the complexity increases unboundedly with the prefix class; this happens already in the two-element case, see [16, Section 7].

We believe that it should also be of interest to study when the bounded-alternation QCSP can be placed in coNP. We have the following conjecture concerning this matter.

Conjecture 4. Let $\mathbf{B}$ be a finite relational structure on a finite signature. If the problem $\mathsf{CSP}(\mathbf{B}^*)$ is in P, then for all $k \geq 1$, the problem Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ is in coNP.

Remark 5. A previous article [15] gives coNP inclusion results for the bounded alternation QCSP in a model that is more general than Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ in that the restriction to the structure is applied only to the existentially quantified variables. The technology developed there should be of help in trying to establish Conjecture 4. The coNP inclusion results presented therein that apply to structures of the form $\mathbf{B}^*$, such as the results on structures preserved by semilattice operations, apply directly to the problems Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ and give evidence for Conjecture 4.

For a structure $\mathbf{B}^*$, when neither the NP upper bound predicted by Conjecture 2 nor the coNP upper bound predicted by Conjecture 4 applies, we believe that the complexity of the problems Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$ should increase unboundedly; perhaps in this case, for each $k \geq 1$, one has Π^p_{2k}-completeness of Π_{2k}-$\mathsf{QCSP}(\mathbf{B}^*)$, a behavior that can be observed in the case of two-element structures [16, Section 7].

5 Unbounded Alternation

In previous articles [14, 17], with the motivation of giving positive QCSP complexity results, we identified and studied properties of algebras called *collapsibility* and *switchability*; switchability is a generalization of collapsibility. Each of these properties implies both the effective PGP property and a reduction from the QCSP to the CSP akin to that given by Theorem 1, but in the general setting of unbounded alternation. Intuitively, the reductions exploit the particular form of generating sets that give the PGP property. Based on this work and as an extension of Conjecture 2 (with Theorem 1), we propose that the PGP property implies a QCSP-to-CSP reduction under unbounded alternation.

Conjecture 5. Let $\mathbf{B}$ be a finite relational structure on a finite signature. If the algebra $\mathbb{A}_{\mathbf{B}}$ has the PGP property, then $\mathsf{QCSP}(\mathbf{B})$ reduces to $\mathsf{CSP}(\mathbf{B}^*)$, and $\mathsf{QCSP}(\mathbf{B})$ is in NP.

Remark 6. Since $\mathbb{A}_{\mathbf{B}} = \mathbb{A}_{\mathbf{B}^*}$ and $\mathsf{QCSP}(\mathbf{B})$ trivially reduces to $\mathsf{QCSP}(\mathbf{B}^*)$, in order to prove Conjecture 5, it suffices to verify it on structures of the form $\mathbf{B}^*$.

As examples showing how the PGP property can be linked to the QCSP complexity properties of Conjecture 5, we revisit two classes of structures whose QCSP is already known to be in NP [28]. We show that each structure therein has an algebra that is *collapsible*, from which it follows that the algebra has the PGP property and the structure enjoys a QCSP-to-CSP reduction [17].

Example 2. (Bipartite graphs) Let $\sigma = \{E\}$ be the signature containing a single binary relation. We consider a structure $\mathbf{B}$ on this signature to be a *bipartite graph* if $E^{\mathbf{B}}$ is symmetric and there exists a partition $B = B_0 \cup B_1$ of the universe

B of $\mathbf{B}$ such that $E^{\mathbf{B}} \subseteq (B_0 \times B_1) \cup (B_1 \times B_0)$. It is known that for each bipartite graph $\mathbf{B}$, the problem $\mathsf{QCSP}(\mathbf{B})$ is in P [28].

Let $w : \{0,1\}^5 \to \{0,1\}$ be the operation that, given a 0/1 tuple of length 5, outputs the unique element in $\{0,1\}$ that occurs 3 or more times in the tuple; the operation w can be thought of as returning the "majority winner", given a tuple of 5 votes. Let mid be the operation defined on non-empty subsets of $\{1,2,3,4,5\}$ that, given a set of size n, ranks the elements in the set as $a_1 < \cdots < a_n$, and outputs the element $a_{\lceil \frac{n}{2} \rceil}$. Intuitively, the operation mid outputs the "middle" element in a given set according to the natural $<$-ranking; on sets of even cardinality, the lower of the two elements in the middle is preferred. For example, $\mathsf{mid}(\{1,4,5\}) = 4$ and $\mathsf{mid}(\{1,2,4,5\}) = 2$.

We now define an operation $p : B^5 \to B$, relative to a partition $B = B_0 \cup B_1$, as follows. Let $s : B \to \{0,1\}$ be the mapping that, given $b \in B$, indicates which side of the partition $B_0 \cup B_1$ the element b lies in; that is, s is defined to be the unique mapping such that for all $b \in B$, it holds that $b \in B_{s(b)}$. Define $p : B^5 \to B$ to be the operation $p(b_1, \ldots, b_5) = b_{p'(b_1,\ldots,b_5)}$, where

$$p'(b_1, \ldots, b_5) = \mathsf{mid}(\{i \mid s(b_i) = w(s(b_1), \ldots, s(b_5))\}).$$

In words, the operation p computes the set of coordinates among $\{1, \ldots, 5\}$ that support the majority winner of the given elements (under s); it then takes the middle coordinate in that set, and projects the given tuple onto that coordinate. The operation p is clearly idempotent.

We show that p is a polymorphism of any bipartite graph $\mathbf{B}$. Suppose that $(b_1, b'_1), \ldots, (b_5, b'_5) \in E^{\mathbf{B}}$. We want to show that $(p(b_1, \ldots, b_5), p(b'_1, \ldots, b'_5)) \in E^{\mathbf{B}}$. We claim that $p'(b_1, \ldots, b_5) = p'(b'_1, \ldots, b'_5)$, which suffices. Since $\mathbf{B}$ is bipartite, for each i (ranging from 1 to 5), we have $s(b_i) \neq s(b'_i)$. It follows that $w(s(b_1), \ldots, s(b_5)) \neq w(s(b'_1), \ldots, s(b'_5))$ and that

$$\{i \mid s(b_i) = w(s(b_1), \ldots, s(b_5))\} = \{i \mid s(b'_i) = w(s(b'_1), \ldots, s(b'_5))\},$$

from which the claim follows.

Collapsibility is a property on algebras which was introduced in the study of the QCSP; collapsibility of an algebra $\mathbb{A}_{\mathbf{B}}$ implies a reduction from $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$ [14]. For an operation $f : A^k \to A$, a coordinate $i \in \{1, \ldots, k\}$, and an element $a \in A$, let $f_i^a : A^{k-1} \to A$ be the operation obtained by f by fixing the ith argument of f to be the element a. It has been shown [14, Lemma 5.13] that an algebra is collapsible if it has an idempotent operation $f : A^k \to A$ and there exists an element $a \in A$ such that each of the k operations f_i^a is surjective. We use this fact to prove the collapsibility of $\mathbb{A}_{\mathbf{B}}$ when $\mathbf{B}$ is a bipartite graph, thus showing that for any bipartite graph $\mathbf{B}$, it holds that $\mathsf{QCSP}(\mathbf{B})$ reduces to $\mathsf{CSP}(\mathbf{B}^*)$. Fix an element $c \in B_0$. We show that each of the 5 operations $p_1^c, \ldots, p_5^c$ is surjective; by the cited lemma, this gives the desired reduction.

We first consider the case $i \neq 3$; in this case, we claim that p_i^c is idempotent, that is, for all $b \in B$, it holds that $p_i^c(b, b, b, b) = b$. The value $p_i^c(b, b, b, b)$ is equal to the value of p on the 5-tuple t that is equal to c at coordinate i, and equal to b elsewhere. We have that $p'(t)$ is equal to $\mathsf{mid}(\{1, 2, 3, 4, 5\}) = 3$ or

$\mathsf{mid}(\{1,2,3,4,5\}\setminus\{i\})$ depending on whether or not $s(b)=0$; in either situation, $p'(t)$ is not equal to i, and the claim follows.

We now consider the case $i=3$. Let $b\in B$ be an element; we want to show that b is in the image of p_3^c. If $s(b)=1$, then we show that $p_3^c(b,b,b,b)=b$. We have $p_3^c(b,b,b,b)=p(b,b,c,b,b)$. Observe that the value $p'(b,b,c,b,b)$ is equal to $\mathsf{mid}(\{1,2,4,5\})=2$, so $p(b,b,c,b,b)=b$. If $s(b)=0$, we argue as follows. Fix d to be an element of B_1. We show that $p_3^c(b,b,b,d)=b$. We have $p_3^c(b,b,b,d)=p(b,b,c,b,d)$. The value $p'(b,b,c,b,d)$ is equal to $\mathsf{mid}(\{1,2,3,4\})=2$, so $p(b,b,c,b,d)=b$. □

Example 3. (Disconnected structures) Let us say that a structure $\mathbf{B}$ on signature σ is *disconnected* if there exists a partition $B_0\cup B_1$ of B composed of two non-empty sets such that for each symbol $R\in\sigma$ and for each tuple $(b_1,\dots,b_k)\in R^{\mathbf{B}}$, it holds that either $\{b_1,\dots,b_k\}\subseteq B_0$ or $\{b_1,\dots,b_k\}\subseteq B_1$.

Let $\mathbf{B}$ be a disconnected structure. We make use of the operations defined in the previous example (Example 2), but assume now that they are with respect to a partition $B=B_0\cup B_1$ that witnesses the disconnectivity of $\mathbf{B}$. We show that the operation $p:B^5\to B$ is a polymorphism of $\mathbf{B}$. By the discussion on collapsibility in the previous example, this yields that $\mathsf{QCSP}(\mathbf{B})$ reduces to $\mathsf{CSP}(\mathbf{B}^*)$. Let $(b_1^1,\dots,b_k^1),\dots(b_1^5,\dots,b_k^5)$ be tuples in a relation $R^{\mathbf{B}}$. We want to show that $(p(b_1^1,\dots,b_1^5),\dots,p(b_k^1,\dots,b_k^5))\in R^{\mathbf{B}}$. We claim that $p'(b_1^1,\dots,b_1^5)=\dots=p'(b_k^1,\dots,b_k^5)$, which suffices. For each i (from 1 to 5), we have $s(b_1^i)=\dots=s(b_k^i)$. For values of j ranging from 1 to k, the values $w(s(b_j^1),\dots,s(b_j^5))$ are all equal, and the sets $\{i\mid s(b_j^i)=w(s(b_j^1),\dots,s(b_j^5))\}$ are all equal; the claim follows. □

Remark 7. Completing the complexity classification of the QCSP on undirected graphs, which was initiated in [28], is an open issue. A complementary objective that we believe could be of interest is to understand which algebras coming from graphs have the PGP.

As an analog of Conjecture 3 in the unbounded alternation setting, we conjecture that here, lack of the PGP property implies PSPACE-completeness of the QCSP. Again, we form a conjecture only on structures of the form $\mathbf{B}^*$.

Conjecture 6. Let $\mathbf{B}$ be a finite relational structure on a finite signature. If the algebra $\mathbb{A}_{\mathbf{B}}$ does not have the PGP property, then the problem $\mathsf{QCSP}(\mathbf{B}^*)$ is PSPACE-complete.

Remark 8. A non-trivial PSPACE-completeness result for a problem $\mathsf{QCSP}(\mathbf{B})$ can be found in [6, Section 6.2]. In particular, such a complexity result is given for certain structures preserved by semilattices without an identity element (recall Example 1). Identifying general sufficient conditions for the PSPACE-hardness of $\mathsf{QCSP}(\mathbf{B})$ is an issue for future research.

Our conjectures focus on structures having constants, that is, structures having the form $\mathbf{B}^*$. We do not yet dare form conjectures on general structures! On

a structure $\mathbf{B}^*$, when one has a reduction from $\mathsf{QCSP}(\mathbf{B}^*)$ to $\mathsf{CSP}(\mathbf{B}^*)$, these two problems are interreducible and have the same complexity (as $\mathsf{CSP}(\mathbf{B}^*)$ is a special case of and reduces to $\mathsf{QCSP}(\mathbf{B}^*)$). Note that the conjectures in this section, along with the conjecture that the CSP admits a dichotomy, predict a P/NP-complete/PSPACE-complete trichotomy in the complexity of the problems $\mathsf{QCSP}(\mathbf{B}^*)$: in the presence of the PGP property, the problem $\mathsf{QCSP}(\mathbf{B}^*)$ is predicted to be interreducible with $\mathsf{CSP}(\mathbf{B}^*)$ (Conjecture 5); otherwise, the problem $\mathsf{QCSP}(\mathbf{B}^*)$ is predicted to be PSPACE-complete (Conjecture 6).

In Examples 2 and 3, we saw structures for which collapsibility can be used to give a reduction from $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$. We now show, via a concrete example, that when $\mathbf{B}$ does not have constants, it is possible that there is such a reduction but that $\mathsf{CSP}(\mathbf{B}^*)$ does *not* characterize the complexity of $\mathsf{QCSP}(\mathbf{B})$: in passing from the problem $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$, one can see a jump in complexity.

Example 4. (Jump from $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$) Let σ be the signature $\{S\}$ where S is of arity 3. Let $\mathbf{A}$ be the structure on σ with universe $\{0,1\}$ and where $S^{\mathbf{A}} = \{(a,b,c) \in \{0,1\}^3 \mid a = b \text{ or } b = c\}$. Let $\mathbf{B}$ be the structure on σ with universe $\{0,1,2\}$ and where $S^{\mathbf{B}} = S^{\mathbf{A}} \cup \{(2,2,2)\}$. The structure $\mathbf{B}$ is clearly disconnected via the partition $B = \{0,1\} \cup \{2\}$, and hence the discussion in Example 3 implies that there is a reduction from $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$. We show that the problem $\mathsf{QCSP}(\mathbf{B})$ is polynomial-time decidable, but the problem $\mathsf{CSP}(\mathbf{B}^*)$ is NP-complete.

We first show that the problem $\mathsf{CSP}(\mathbf{B}^*)$ is NP-complete. We reduce from the problem $\mathsf{CSP}(\mathbf{A}^*)$, which is NP-complete by Schaefer's theorem. Given an instance $\Phi = \exists v_1 \ldots \exists v_n \phi$ of $\mathsf{CSP}(\mathbf{A}^*)$, where ϕ is quantifier-free, the reduction creates the instance Φ' defined as

$$\exists v_1 \ldots \exists v_n \exists c_0 \exists c_1 (\phi \wedge R_0(c_0) \wedge R_1(c_1) \wedge \bigwedge_{i=1}^{n} S(c_0, v_i, c_1)).$$

In this latter instance, the variables c_0 and c_1 are assumed to be fresh variables that do not occur among $v_1, \ldots, v_n$; via the formulas $R_0(c_0)$ and $R_1(c_1)$, they are forced to the values 0 and 1. The formulas $S(c_0, v_i, c_1)$ force each variable v_i to take on the value 0 or 1. (Note that this forcing can also be done with just one of the constants c_0, c_1; for instance, the formula $S(c_0, c_0, v_i)$ also forces the variable v_i to take on the value 0 or 1.) With these facts in mind, it is readily verified that Φ is true on $\mathbf{A}^*$ if and only if Φ' is true on $\mathbf{B}^*$.

We now show that $\mathsf{QCSP}(\mathbf{B})$ is polynomial-time decidable. Given an instance Φ of $\mathsf{QCSP}(\mathbf{B})$, let G_Φ be the undirected graph whose vertices are the variables of Φ and where an edge $\{u, v\}$ is present if and only if u and v occur together in a predicate application. For each connected component C of G_Φ, we define C' to be the set obtained by removing the variable that is quantified first in C. We claim that Φ is true if and only if the following condition holds: for each connected component C of G_Φ, the set C' contains only existentially quantified variables. This suffices, since the condition is readily checkable in polynomial time.

We verify the claim as follows. We view Φ as a game between two players, universal and existential, that set the respectively quantified variables in the order given by the quantifier prefix; the universal player tries to falsify the quantifier-free part ϕ of Φ, whereas the existential player tries to satisfy ϕ. If the condition holds, then for each connected component C, by setting all of the variables in C' appropriately, the existential player can guarantee that all variables in C are set to the same value. This suffices to satisfy ϕ, since all three constant tuples $(0,0,0),(1,1,1),(2,2,2)$ are contained in $S^{\mathbf{B}}$. If the condition does not hold, then there exists a connected component C such that there is a universally quantified variable y in C'. Let v be the variable in C but not in C', and let us call $\{0,1\}$ and $\{2\}$ the *blocks* of B. The universal player can then guarantee that y is set to a value that is in a different block from the value given to v. This suffices to spoil ϕ, since to satisfy ϕ it must be that all variables in C are set to values in the same block. □

From the perspective of this example, the approach of giving a reduction from $\mathsf{QCSP}(\mathbf{B})$ to $\mathsf{CSP}(\mathbf{B}^*)$ is not as fine a tool as one would like for understanding the complexity of $\mathsf{QCSP}(\mathbf{B})$: while this approach gives an NP upper bound on $\mathsf{QCSP}(\mathbf{B})$, the problem $\mathsf{CSP}(\mathbf{B}^*)$ does not always yield the more detailed information of whether or not $\mathsf{QCSP}(\mathbf{B})$ is in P. Broadly speaking, the development of tools for understanding the QCSP on general structures (by which is meant structures not necessarily having constants) would be very welcome.

Acknowledgements. Simone Bova and Moritz Müller provided numerous helpful suggestions. Johan Thapper supplied pointed feedback and many corrections that induced both revision of this article as well as further meditation. I also thank Andrei Bulatov, Florent Madelaine, Barny Martin, and Peter Mayr for useful discussions, and Matt Valeriote for his support. Finally, I express gratitude to Dexter Kozen, who supervised my doctoral thesis on quantified constraint satisfaction, which thesis is the ultimate origin of these meditations. The author is supported by the Spanish program "Ramon y Cajal" and MICINN grant TIN2010-20967-C04-02.

References

1. Barto, L., Kozik, M.: Constraint satisfaction problems of bounded width. In: Proceedings of FOCS 2009 (2009)
2. Berman, J., Idziak, P., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Varieties with few subalgebras of powers. Transactions of the American Mathematical Society 362(3), 1445–1473 (2010)
3. Bodirsky, M., Chen, H.: Quantified equality constraints. SIAM Journal on Computing 39(8), 3682–3699 (2010)
4. Bodirsky, M., Kára, J.: The complexity of temporal constraint satisfaction problems. Journal of the ACM 57(2) (2010)
5. Bodirsky, M., Nešetřil, J.: Constraint satisfaction with countable homogeneous templates. Journal of Logic and Computation 16(3), 359–373 (2006)

6. Börner, F., Bulatov, A.A., Chen, H., Jeavons, P., Krokhin, A.A.: The complexity of constraint satisfaction games and QCSP. Information and Computation 207(9), 923–944 (2009)
7. Bova, S., Chen, H., Valeriote, M.: Generic Expression Hardness Results for Primitive Positive Formula Comparison. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 344–355. Springer, Heidelberg (2011)
8. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the Complexity of Constraints using Finite Algebras. SIAM Journal on Computing 34(3), 720–742 (2005)
9. Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the ACM (JACM) 53 (2006)
10. Bulatov, A.A.: The Complexity of the Counting Constraint Satisfaction Problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 646–661. Springer, Heidelberg (2008)
11. Chen, H.: Quantified Constraint Satisfaction and 2-Semilattice Polymorphisms. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 168–181. Springer, Heidelberg (2004)
12. Chen, H.: Quantified Constraint Satisfaction, Maximal Constraint Languages, and Symmetric Polymorphisms. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 315–326. Springer, Heidelberg (2005)
13. Chen, H.: Quantified Constraint Satisfaction, Maximal Constraint Languages, and Symmetric Polymorphisms. In: Electronic Colloquium on Computational Complexity, ECCC (2005)
14. Chen, H.: The Complexity of Quantified Constraint Satisfaction: Collapsibility, Sink Algebras, and the Three-Element Case. SIAM Journal on Computing 37(5), 1674–1701 (2008)
15. Chen, H.: Existentially restricted quantified constraint satisfaction. Information and Computation 207(3), 369–388 (2009)
16. Chen, H.: A rendezvous of logic, complexity, and algebra. ACM Computing Surveys 42(1) (2009)
17. Chen, H.: Quantified constraint satisfaction and the polynomially generated powers property. Algebra Universalis 65, 213–241 (2011)
18. Cohen, D.A., Creed, P., Jeavons, P.G., Živný, S.: An Algebraic Theory of Complexity for Valued Constraints: Establishing a Galois Connection. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 231–242. Springer, Heidelberg (2011)
19. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. SIAM Journal on Computing 28, 57–104 (1999)
20. Hyde, J.T., Louglin, N.J., Quick, M., Ruškuc, N., Wallis, A.R.: On the growth of generating sets for direct powers of semigroups. Semigroup Forum
21. Idziak, P., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Tractability and learnability arising from algebras with few subpowers. SIAM J. Comput. 39(7), 3023–3037 (2010)
22. Jeavons, P., Cohen, D., Cooper, M.: Constraints, consistency, and closure. Artificial Intelligence 101(1-2), 251–265 (1998)
23. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200, 185–204 (1998)
24. Jeavons, P.G., Cohen, D.A., Gyssens, M.: Closure Properties of Constraints. Journal of the ACM 44, 527–548 (1997)

25. Jonsson, P., Kuivinen, F., Nordh, G.: Max Ones generalized to larger domains. SIAM Journal on Computing 38, 329–365 (2008)
26. Larose, B., Zádori, L.: Bounded width problems and algebras. Algebra Universalis 56(3-4), 439–466 (2007)
27. Larose, B., Tesson, P.: Universal algebra and hardness results for constraint satisfaction problems. Theoretical Computer Science 410(18), 1629–1647 (2009)
28. Martin, B., Madelaine, F.: Towards a Trichotomy for Quantified *H*-Coloring. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 342–352. Springer, Heidelberg (2006)
29. Meier, D., Wiegold, J.: Growth sequences of finite groups V. Journal of the Australian Mathematical Society (Series A) 31, 374–375 (1981)
30. Quick, M., Ruškuc, N.: Growth of generating sets for direct powers of classical algebraic structures. Journal of the Australian Mathematical Society 89, 105–126 (2010)
31. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of STOC 1978, pp. 216–226 (1978)
32. Wiegold, J.: Growth sequences of finite groups. Journal of the Australian Mathematical Society (Series A) 17, 133–141 (1974)
33. Wiegold, J.: Growth sequences of finite groups II. Journal of the Australian Mathematical Society (Series A) 20(2), 225–229 (1975)
34. Wiegold, J.: Growth sequences of finite groups III. Journal of the Australian Mathematical Society (Series A) 25(2), 142–144 (1978)
35. Wiegold, J.: Growth sequences of finite groups IV. Journal of the Australian Mathematical Society (Series A) 29, 14–16 (1980)
36. Wiegold, J.: Growth sequences of finite semigroups. Journal of the Australian Mathematical Society (Series A) 43, 16–20 (1987)

The Compass That Steered Robotics

Bruce R. Donald

Department of Computer Science
Duke University
Department of Biochemistry
Duke University Medical Center
www.cs.duke.edu/brd

Abstract. Robotics researchers will be aware of Dexter Kozen's contributions to algebraic algorithms, which have enabled the widespread use of the theory of real closed fields and polynomial arithmetic for motion planning. However, Dexter has also made several important contributions to the theory of information invariants, and produced some of the most profound results in this field. These are first embodied in his 1978 paper *On the Power of the Compass*, with Manuel Blum. This work has had a wide impact in robotics and nanoscience.

Starting with Dexter's insights, robotics researchers have explored the problem of determining the information requirements to perform robot tasks, using the concept of *information invariants*. This represents an attempt to characterize a family of complicated and subtle issues concerned with measuring robot task complexity.

In this vein, several measures have been proposed [14] to measure the information complexity of a task: (a) How much internal state should the robot retain? (b) How many cooperating robots are required, and how much communication between them is necessary? (c) How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task? (d) How much information is provided by sensors? and (e) How much computation is required by the robot? We have considered how one might develop a kind of "calculus" on (a) – (e) in order to compare the power of sensor systems analytically. To this end, information invariants is a theory whereby one sensor can be "reduced" to another (much in the spirit of computation-theoretic reductions), by adding, deleting, and reallocating (a) – (e) among collaborating autonomous robots. As we show below, this work steers using Dexter's compass.

1 The Power of the Compass

In 1978, Blum and Kozen wrote a ground-breaking paper on maze-searching automata [2,38]. This chapter is devoted to a discussion of their results, *On The Power of the Compass* [2], and we interpret their results in the context of autonomous mobile robots and information invariants.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 50–65, 2012.

1.1 Notation

In this chapter, I use (a), (b), (c), ... to denote *Resources*, such as internal state, number of robots, external state, and so forth (see **Abstract** for a complete list). The numbers (1), (2), (3) denote a list of key results from Dexter's paper [2], which are introduced in Section 1.2. Starred roman numerals I*, II*, III*, ... denote techniques in information invariants theory (such as *Reduction*, *Transformation*, *Universal Reduction*, etc.); these are described in Section 3.3. Small roman numerals (i), (ii) denote resources for information invariants in massively-parallel distributed manipulation and nanoscience (Section 3.2).

1.2 The Scales Fall from My Eyes

From 1987-1997, I taught at Cornell, just down the hall from Dexter. My health was excellent. Every morning I drank Pepsi before teaching large undergraduate programming lectures. Each afternoon I drank espresso and wrote papers, while watching the sun set over Lake Cayuga from my office (which was the largest lair, with the best view, in Upson Hall). In the evenings I would eat dinner with Dan Huttenlocher or Ramin Zabih, and at night I played in Dexter's band, *The Steamin' Weenies.* I tended a large flock of enthusiastic graduate students and post-docs working on robotics. In 1990, my student Jim Jennings and I posed the following:

Question 1. **[35]** "Let us consider a rational reconstruction of mobile robot programming. There is a task we wish the mobile robot to perform, and the task is specified in terms of external (e.g., human-specified) perceptual categories. For example, these terms might be "concepts" like *wall*, *door*, *hallway*, or *Professor Hopcroft.* The task may be specified in these terms by imagining the robot has *virtual* sensors which can recognize these objects (e.g., a wall sensor) and their "parameters" (e.g., length, orientation, etc.). Now, of course the physical robot is not equipped with such sensors, but instead is armed with certain *concrete* physical sensors, plus the power to retain history and to compute. The task-level programming problem lies in implementing the virtual sensors in terms of the concrete robot capabilities. We imagine this implementation as a tree of computation, in which the vertices are control and sensing actions, computation, and state retention. A particular kind of state consists of geometric constructions; in short, we imagine the mobile robot as an automaton, connected to physical sensors and actuators, which can move and interrogate the world through its sensors while taking notes by making geometric constructions on "scratch paper." But what should these constructions be? What program runs on the robot? How may these computation trees be synthesized?"

Let us consider this question of state. Suppose the robot is given a particular task. To accomplish this task, what should the robot record on its scratch paper? What is necessary and sufficient? In robotics, necessity has rarely been addressed. Sufficiency has been addressed but the bounds are extremely loose. Specifically: in robotics, the answer for sufficiency is frequently either "nothing" (i.e., the robot is reactive, and should not build any representations), or "a map"

(namely, the robot should build a geometric model of the entire environment). In particular, even schemes such as [41] require a worst-case linear amount of storage (in the geometric complexity n of the environment). Can one do better? Is there a sufficient representation that is between 0 and $O(n)$?

This seemed like a great question to work on. Dexter's office was three doors down down the hall (*hear that, robot?*), so we kicked it around. Dexter mentioned he had "some results" on this problem, and gave me a copy of his 1978 paper.

"Some results" turned out to be a considerable understatement. His paper laid out the foundations for the field, posing and solving its first and most fundamental problems. As I read his paper, my excitement grew with each page. Blum and Kozen provided precise answers to these questions in the setting of theoretical, situated automata. The results provide substantial insight into the Question 1 above. His paper had a profound impact on my work [14].

This chapter didactically adopts the rhetorical "we" to compactly interpret Dexter's results. We define a *maze* to be a finite, two-dimensional obstructed checkerboard. A finite automaton (DFA) in the maze may, in addition to its automaton transitions, transit on each move to an adjacent unobstructed square in the N, S, E, or W direction. We say an automaton can *search* a maze if eventually it will visit each square. It need not halt, and it may revisit squares. Hence, this kind of "searching" is the theoretical analog of the "exploration" task that many modern mobile robots are programmed to perform. However, note that in this entire section there is no control or sensing uncertainty.

We can consider augmenting an automaton with a single *counter*; using this counter it can record state. Two counters would not be an interesting enhancement, because then we obtain the power of a Turing machine.[1] The distinction is that a DFA with two counters is as powerful as a Turing machine (which can make a linear-sized map) so in some sense this augmentation of a DFA is naïve, or trivial. We wish to address the the question of whether or not there exists a DFA space augmentation that lies in between 'nothing' and a 'full Turing machine.' In this manner we can explore whether or not tasks can be accomplished without making a linear-sized map. The question can be nicely explored by asking: *what is the power of giving the DFA a single counter?*

We say two (or more) automata *search a maze together* as follows. The automata move synchronously, in lock-step, but at each step the DFAs can perform

[1] A *counter* is like a register. A DFA with a *counter* can keep a count in the register, increment or decrement it, and test for zero. A single counter DFA (introduced by [30] in 1966) can be viewed as a special kind of push-down (stack) automaton (PDA) that has only one stack symbol (except for a top of the stack marker). This means we should not expect a single-counter machine to be more powerful than a PDA, which, in turn, is considerably weaker than a Turing machine (see, eg., [33, Ch. 5]). The proof that a two-counter DFA can simulate a Turing machine was first given by Papert and McNaughton in 1961 [43] but shorter proofs are now given in many textbooks, for example, see [33, Thm. 7.9]. However, our distinction of *one counter vs. two counters* is motivated by theory, and is mathematical rather than practical. In practice, one would not equip a robot with two counters to simulate a Turing machine, because the simulation is not efficient.

different internal state transitions and step in different directions on the maze. This synchronization could be effected using global control, or with synchronized clocks. When two automata land on the same square, each transmits its internal state to the other.

Finally, we may *externalize* and *distribute* the state. Instead of a counter, we may consider equipping an automaton with *pebbles*, which it can drop and pick up. Each pebble is uniquely identifiable to any automaton in the maze. On moving to a square, an automaton senses what pebbles are on the square, plus what pebbles it is carrying. It may then drop or pick up any pebbles.

Hence, a pure automaton is a theoretical model of a "reactive," robot-like creature. (Many simple physical robot controllers are based on DFA's). The exchange of state between two automata models local communication between autonomous robots. The pebbles model the "beacons" often used by mobile robots, or, more generally, the ability to side-effect the environment (as opposed to the robot's internal state) in order to perform tasks. Finally, the single counter models a limited form of internal state (storage). It is much more restrictive than the tape of a Turing machine. Quantifying communication between collaborating mobile robots is a fundamental information-theoretic question. In manipulation, the ability to structure the environment through the actions of the robot (see, eg, [13,14,23,48]) or the mechanics of the task (see, eg,. [42]) is a fundamental paradigm. How do these techniques compare in power?

We call automata with these extra pebbles or counters *enhanced*, and we will assume that automata are not enhanced unless noted. All automata are deterministic, and there is no randomization unless explicitly noted. Given these assumptions, Blum and Kozen demonstrate the following results. First, they note a result of Budach that a single automaton cannot search all mazes.[2] Next they prove the following:

1. There are two (unenhanced) automata that together can search all mazes.
2. There is a two-pebble automaton that can search all mazes.
3. There is a one-counter automaton that can search all mazes.

We will show below that these results are crisp information invariants. It is clear that a Turing machine could build (a perfect) map of the maze, that would be linear in the size of the maze. This they term the *naïve linear-space algorithm.* This is the theoretical analog of most map-building mobile robots—even those that build "topological" maps still build a linear-space geometric data structure on their "scratch paper." But (3) implies that there is a *log-space* algorithm to search mazes—that is, using only an amount of storage that is logarithmic in the complexity of the world, the maze can be searched. Why? Here is the idea: First, [2] show how to write a program whereby an unenhanced DFA can traverse the boundary of any single connected component of obstacle squares. Now, suppose the DFA could "remember" the southwesternmost corner (in a lexicographic order) of the obstacle. Next, [2] show how all the free space can then be systematicically searched. It suffices for a DFA with a single counter to record the y-coordinate $y_{\min}$ of this corner. We now imagine simulating this

[2] See [2] for references.

algorithm (as efficiently as possible) using a Turing machine, and we measure the bit-complexity. If there are n free squares in the environment then $y_{\min} \leq n$, and the algorithm consumes $O(\log n)$ bits of storage. For details, see [2]. This is a precise answer to part of our Question 1.

However, the results (1-3) also demonstrate interesting information invariants. (1) = (2) demonstrates the equivalence (in the sense of information) of beacons and communication. Hence, side-effecting the environment is equivalent to collaborating with an autonomous co-robot. In other words, the augmentations to the DFA of (1) and (2) are equivalent in power, in that either (1) or (2) allows the robot to accomplish the maze-searching task. The equivalence of (1) = (2) = (3) suggests an equivalence (in this case) and a tradeoff (in general) between communication, state, and side-effecting the environment. We credit [2] with these founding examples of information invariants.

1.3 The Power of Randomization

Michael Erdmann's Ph.D. thesis was an investigation of the power of randomization in robotic strategies [26]. The idea is similar to that of randomized algorithms—by permitting the robot to randomly perturb initial conditions (the environment), its own internal state, or to randomly choose among actions, one may enhance the performance and capabilities of robots, and derive probabilistic bounds on expected performance.[3] This lesson should not be lost in the context of the information invariants above. For example, as Erdmann points out, one finite automaton can search any maze if we permit it to randomly select among the unobstructed directions. The probability that such an automaton will eventually visit any particular maze square is 1. Randomization also helps in finite 3D mazes (see Section 1.4 for more on the problems that deterministic (as opposed to randomized) finite automata have in searching 3D mazes), although the expected time for the search increases some.

These observations about randomizing automata can be even extended to *unbounded* mazes (the mazes we have considered so far in this chapter are finite). However, in a 2D unbounded maze, although the automaton will eventually visit any particular maze square with probability 1, the expected time to visit it is infinite. In 3D, however, things are worse: in 3D unbounded mazes, the probability that any given "cube" will be visited drops from 1 to about 0.37.

1.4 What Does a Compass Give You?

Thus we have given precise examples of information invariants for *tasks* (or for one task, namely, searching, or "exploration.") However, it may be less clear what the information invariants for a *sensor* would be. Again, Blum and Kozen provide a fundamental insight. We motivate their result with the following

[3] While the power of randomization has long been known in the context of algorithms for maze exploration, Erdmann was able to lift these results to the robotics domain. In particular, one challenge was to consider continuous state spaces (as opposed to graphs).

Question 2. Suppose we have two mobile robots, named Tommy and Lily, configured as described in [14]. Suppose we put a flux-gate magnetic compass on Lily (but not on Tommy). How much more "powerful" has Lily become? What tasks can Lily now perform that Tommy cannot?

Now, any robot engineer knows compasses are useful. But what we want in answer to Question 2 is a precise, provable answer. Happily, in the case where the compass is relatively accurate,[4] [2] provide the fundamental insight:

Consider an automaton (of any kind) in a maze. Such an automaton effectively has a compass, since it can tell directions N,S,E,W apart. That is, on landing on a square, it can interrogate the neighboring N,S,E,W squares to find out which are unobstructed, and it can then accurately move one square in any unobstructed compass direction.

By contrast, consider an automaton in a *graph* (that need not be a maze). Such an automaton has no compass; on landing on a vertex, there are some number $g \geq 0$ of unordered edges leading to "free" other vertices, and the automaton must choose one.

Hence, as Blum and Kozen point out, *"Mazes and regular planar graphs appear similar on the surface, but in fact differ substantially. The primary difference is that an automaton in a maze has a* compass*: it can distinguish N,S,E,W. A compass can provide the automaton with valuable information, as shown by the second of our results"* [2]. Now, assume all automata are deterministic, and no randomization is permitted. Recall result (1) in Section 1.2: *(1) There are two (unenhanced) automata that together can search all mazes.* Blum and Kozen show, that in contrast to (1), no two automata together can search all finite planar cubic graphs (in a *cubic* graph, all vertices have degree $g = 3$). They then prove no three automata suffice. Later, Kozen showed that four automata do not suffice [38]. Moreover, if we relax the planarity assumption but restrict our cubic graphs to be 3D mazes, it is known that no finite set of finite automata can search all such finite 3D mazes [3]!

Hence, [2,38] provide a lower bound to the question, "What information does a compass provide?" We close by mentioning that in the flavor of Section 1.3, there is a large literature on randomized search algorithms for graphs. As in Section 1.3, randomization can improve the capability and performance of the search automata.

2 Measuring Information Invariants

Blum and Kozen gave us the basic tools and concepts behind information invariants. We illustrated by example how such invariants can be analyzed and derived. We made a conceptual connection between information invariants and trade-offs. Tradeoffs also arise naturally in *kinodynamic settings* [24], in which

[4] In considering how an accurate sensor can aid a robot in accomplishing a task, Dexter's methodology anticipates, as it were, Erdmann's work on developing "minimal" sensors [27].

performance measures, planning complexity, and robustness (in the sense of resistance to control uncertainty) are traded-off [24,21,22]. We noted that Erdmann's invariants are of this ilk [26]. More generally, in optimization problems (shortest path, fastest path, etc.) it is natural to define trade-offs using these performance measures (e.g., path-length, -time, or -cost) as a kind of common currency. Indeed, such trade-offs form the basis of online algorithms and polynomial-time approximation schemes.

However, *without* a performance (cost) measure, it is substantially more difficult to develop information invariants. This is where the beauty of Dexter's approach is evident. Measures of *information complexity* are fundamentally different from *performance measures.* Our interest in this chapter lies in the former (for more on performance measures see [14], and [24]).

Here are some measures of the information complexity of a robotic task: *(a) How much internal state should the robot retain? (b) How many cooperating robots are required, and how much communication between them is necessary?* and *(c) How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task?* Examples of these categories include: (a) space considerations for computer memory, (b) local line of sight communication such as infra-red (IR) communication between collaborating autonomous mobile robots, and (c) dropable beacons. With regard to (a), we note that, of course, memory chips are cheap, but in the mobile robot design space, most investigations seem to fall at the ends of the design spectrum. For example, (near) reactive systems use (almost) no state, while "map builders" and model-based approaches use a very large (linear) amount. Natarajan [44] considered an invariant complexity measure analogous to (b), namely the number of robot "hands" required to perform an assembly task. This quantifies the interference kinematics of the assembly task, and assumes global synchronous control. With regard to (c), one easily-imagined physical realization consists of coded IR beacons; however, "external" side-effects could be as exotic as chalking notes on the environment (as parking police do on tires), or assembling a collection of objects into a configuration of lower "entropy" (and hence, greater information). *Calibration* is an important form of external state (or, more generally a way to synchronize internal state, robot configuration, and external state), which we explore in [14].

Dexter proved automata-theoretic results to explore invariants that trade-off internal state, communication, and external state. His work first concentrates on information invariants for *tasks.* It then shows how information invariants for *sensors* can be integrated into the discussion. In particular, Dexter gave a precise way to measure the information that a compass gives an autonomous mobile robot. Remarkably, trading off the measures (a)-(c) proved sufficient to quantify the information a compass supplies!

The compass invariant illustrates the kind of result that we wish to prove for more general sensors. Thus, we add a measure to quantify the information provided by sensors. To push this framework further, we had to introduce additional machinery to include two additional important measures of the

information complexity of a robotic task: *(d) How much information is provided by sensors?*, and *(e) How much computation is required of the robot?* In [14], we described how one might develop a kind of "calculus" on measures (a) – (e) in order to compare the power of sensor systems analytically. To this end, we developed a theory whereby one sensori-computational system can be "reduced" to another (much in the spirit of computation-theoretic reductions), by adding, deleting, and reallocating (a) – (e) among collaborating autonomous robots.

3 Impact on Robotics and Nanoscience

Dexter's ideas and their offspring in the information invariants literature [14] have had a wide impact on robotics in general and microrobotics in particular. I give three examples. Videos of these implemented robotic systems can be found online at: [15,16].

3.1 Microscale Assembly

[20] describe top-down microassembly using groups of non-holonomic, highly under-actuated micro-robots. A detailed description of an individual robot can be found in [19]. Such an assembly would be rather easy at the macroscopic scale but the individual robots are about 200 by 60 microns in size, making control and assembly challenging. These robots demonstrate information invariant trade-offs in terms of control and design. The control is encoded in the power-delivery signal, which must be demultiplexed by the robots. All the robots receive the same global power delivery and control signal but respond differently not only because of their different internal states, but also due to engineered differences in their physics.

Since even a single robot [19] is under-actuated and non-holonomic, information invariants-based design was necessary to prove global controllability. The robots in the papers and videos [20] exhibit an unprecedented degree of individual control, for things that are so tiny. The robots are intentionally simple in design to minimize their individual size, and groups of such microrobots are highly underactuated when directed using a broadcast control signal. The control algorithms reconfigure this highly underactuated n-microrobot system using a non-holonomic control scheme. This was the first example of parallel (simultaneous) operation and cooperation of multiple untethered microelectromechanical system (MEMS) microrobots.

3.2 Provable Constraints on Architecture and Dynamics for Massively Parallel, Distributed Manipulation

Background. We now discuss an interesting application, also in microassembly. Part manipulation is an important but also time-consuming operation in microscale automation. Micro-parts need to be sorted and oriented before assembly. It is a difficult problem to manipulate, orient, singulate, and assemble such parts at the microscale.

One possibility is to use a massively parallel array of distributed microactuators in order to perform distributed manipulation [5,9,6,10,11,49,4]. The microactuators are controlled using programmable force fields. The basic idea is the following: the field is realized on a planar surface on which the part is placed. The forces exerted on the contact surface of the part translate and rotate the part to an equilibrium configuration. The manipulation requires no sensing. Current technology permits the implementation of certain force fields in the microscale with 'ciliary' actuator arrays built in MEMS technology, and in the macroscale with transversely vibrating plates. The flexibility and dexterity that programmable force fields offer has led researchers to investigate the extent to which these fields can be useful. Some work [5,9,6,10,11,49,4,7] analyzes the properties of force fields that are suitable for sensorless manipulation and proposes novel manipulation strategies. These strategies typically consist of temporally discrete sequences of force fields that cascade the parts through multiple equilibria until a desired goal state is reached.

For example, one may develop a sequence of steps (a sequence of vector fields) to orient a polygonal part. Programmable force fields allow us to shift the complexity of parts-feeding from the design of mechanical tracks, filters, and cutouts, to control algorithms and circuitry. No sensors or feeder redesign is required. However, the first designs required control software, a clock, and, to some extent, synchronization between distributed actuators. In three papers [7,9,5], we addressed the information invariants trade-offs in such devices, specifically the trade-off between (i) having a clock and communication for sequencing, versus (ii) using a more complex vector field that obviates the necessity of a clock for synchronization and sequencing [7]. Finally, we showed that, surprisingly, one of the more complex components of the vector field can be implemented by a coupling with the world (gravity) in combination with a relatively simple MEMS array [9,49,5].

Significance and Generalizability. We now discuss the relevance to a general methodology of information invariants for control and manipulation in a distributed setting. Suppose we take the view of an architect seeking to simplify a massively-parallel distributed system, namely our microscale parts feeder. For discussion, we will adapt a perspective that has been profitable in distributed systems, and try to remove the clock from the distributed system (this system comprises the massively parallel microfabricated actuator array, together with its control, communication, and computation). Specifically: typical MEMS arrays for programmable force fields require control lines for programmability, plus a clock to switch between control strategies. In addition, control hardware and software are required, for example in computer(s) connected to the actuator array. Let us ask the 'minimalist' question: *In what ways can the system be simplified?*

One direction to explore is the following: can the clock be removed? Somewhat remarkably, this question proves to be equivalent to the conjecture: *Does there exist a single vector field U in which every part P has exactly one stable equilibrium x_P (up to part symmetry)?* The reason for equivalence is: unless such

a unique equilibrium exists, then a clock will be needed to cascade and collapse the multiple equilibria by switching after some time to a subsequent vector field strategy.

Specifically: If such a 'universal' field exists, part orientation can be effected without sensing and without a clock, achieving a minimal solution in terms of resources. It is surprising that a purely architectural question can reduce to a proving a conjecture about geometric dynamics. Details of the proof can be found in [7]. It also illustrates the interplay of continuous methods to prove bounds from, and on, discrete architectural constraints. This example illustrates information invariants between clock synchronization and vector field complexity. While much work has been done in the complexity of various branches of computational mathematics (algebra, geometry, topology), the complexity of vector fields on manifolds has only recently been considered. Now that these vector fields are a programming paradigm for massively-parallel distributed manipulation, systematic theoretical investigations have born fruit, to prove these counterintuitive and powerful results [5,9,6,10,11,4,7].

Dexter's results on information invariants for multiple cooperating DFAs not only inspired a generation of researchers to work on parallel and distributed robotics, but also showed them how robotics can be approached as a science, with provable resource trade-offs driving a rigorous analysis of complexity, soundness, and completeness. When his approach was understood by roboticists in the 1990s, they were working with (at most) small handfuls of laboriously handcrafted mobile robots (4, 5, or possibly 10). Since simulations were doomed to success, Dexter's work motivated robotics researchers to find a domain where questions of parallel and distributed robotics could be explored *experimentally* for tens of thousands, if not millions of cooperating actuators. MEMS provided an ideal testbed for such theories, since bulk fabrication allows the construction of huge numbers of microactuators (in the same way that IC circuits are fabricated using VLSI). However, the pioneers who moved from robotics to MEMS were explicitly trying to generalize Dexter's results and obtain crisp theoretical information invariants that could be experimentally validated. In some sense, the migration from robotics to nanoscience was a multi-university physics experiment, designed to determine how Dexter's laws of parallel robotics would scale and generalize to massively-parallel distributed manipulation. The fruit of this research is *the theory of programmable vector fields,* which we have reviewed briefly in this chapter.

The theory of programmable vector fields for micro- and nano-scale manipulation has yielded numerous interesting theoretical results and predictions, that have been confirmed by extensive experimental validation [5,9,6,10,11,49,4,7]. The theory grew out of information invariants analysis, and represents a powerful technique for massively-parallel distributed manipulation. The degree of parallelism and distribution in these manipulation tasks is much higher than in other branches of robotics: tens of thousands of microactuators can easily be controlled and coordinated, in sophisticated manipulation tasks, and there is no reason it shouldn't work for millions or billions. There are many theories of

multi-robot and multi-actuator control. There are also theories of manipulation, and sometimes even theories for parallel and distributed manipulation. Typically, even the best of these theories break down as the number of actuators increases. But the algorithmic theory of programmable vector fields for massively-parallel distributed manipulation is the only technique for multi-robot control that becomes more robust and more accurate as the actuators become more numerous, smaller, and denser. And at this point, the algorithms that Dexter's work inspired have been implemented in hardware using silicon, polyimide, and metal, leveraging a dizzying array of 21^{st}-century surface chemistry and nanofabrication technologies. This is no small feat.

3.3 Trade-Offs, Robot Complexity, and Information Invariants

Information invariants as a theory have been used generate and analyze interesting experiments in the field of mobile robotics. For example, in the 1990s this methodology was used in a significant demonstration of a distributed multi-mobile-robot team to push an object into place [23,48]. These explorations into information invariants have had impact on the multi-robot research community. It also led to a careful analysis of the trade-offs in massively-parallel distributed manipulation using microfabricated actuator arrays, described above in Sec. 3.2. Perhaps more important, the work on information invariants in the solution to robot tasks made precise what had previously been only an inchoate notion, namely: that robots can gain information by action or by sensing or by internal state, and that the sources of information are to some extent interchangeable. Of particular power are the method of *sensor reductions* and the construct of *permutation* for reallocating resources [14]. Sensor reductions are analogous to computation-theoretic reductions in that they allow mobile sensor networks to be rigorously compared, and induce a hierarchy of complexity over the class of sensori-computational systems. But because sensor networks are embedded in Whitney stratifications (i.e., composed of differentiable algebraic manifolds), many questions about them can, in principle, be decided computationally. Hence, in contrast to computation-theoretic reductions, the reduction (i.e. complexity) hierarchy of information invariants on sensory networks is effectively computable.

Four key techniques are made possible in the information invariants framework:

- I* Given two sensori-computational systems, we can ask which is more powerful (can one be reduced to the other)?
- II* We can also ask, can one sensori-computational system be transformed into another, and if not, what resources must be added to make it equivalent?
- III* Given a collection of "parts" (resources) and a specification of a sensori-computational system, can the parts be configured to implement that specification?
- IV* Universal reduction: can the components of one sensori-computational system always perform the job of a second sensori-computational system?

It is also remarkable that, again, in principle, all four of these decision problems have been shown to be effectively computable [14]. One of the difficult and challenging aspects of theoretical computer science and structural complexity theory is that the reductions that leverage many theorems must be crafted by humans, since the existence and form of these reductions is not effectively computable. By this we mean the following. Suppose we have two computation-theoretic problems. Can one be reduced to the other? There is no algorithm for deciding this. Instead, a proof must be constructed by a human. The contrast we wish to make is that in the domain of parallel and distributed robotics, there is an algorithm to decide whether or not one sensori-computational system can be reduced to another. Moreover, the algorithm is constructive and the reduction can be effectively computed.

Hence, distributed and parallel robotics provides a domain with a rich complexity hierarchy, in which, unlike in the general theory of computation, reductions between sensori-computational systems can be effectively computed. The ability to compute these reductions comes directly from information invariants, namely from the embedding of the physical robot systems into real semialgebraic sets. Apart from its importance to robotics, this means that some difficult questions of hierarchy, equivalence, hardness, and classification, all of which interest theoretical computer science, can be explored in a more tractable alternative domain.

Despite this progress, there is much to be done in developing and applying the information invariants theory. First, the theory is perhaps most powerful at quantifying trade-offs between communication and sensing. For example, the machinery can be used to eliminate explicit communication between robots in order to allow them to communicate through the task [23,48]. The information invariants mechanism uses a hierarchy of reductions (that satisfy 'graded transitivity') to compare the power of sensori-computational systems and to compute transformations between them [14]. However, the theory is still not fully elaborated for manipulation tasks and action/motion in general. In its present state, the information invariants theory can apply to a sensory system which is embedded like a graph, or whose vertices are constrained to lie in sets within a configuration space. While clearly this represents a kind of dynamics or motion, the theory does not exploit the motion as encoded in trajectories, and the mechanics of manipulation is not explicitly represented.

For this reason, [14,5,9,6,23,48,10,11,49,4,7,45,19,20] studied, by specific examples, a series of challenging distributed manipulation problems that would foreground the issues of distribution, parallelism, manipulation, and mechanics (this is embodied in our work on massively-parallel distributed manipulation using microfabricated actuator arrays, and subsequent other MEMS microrobot work). In this domain, the scale of the parallelism is large and therefore an appealing test case. The manipulation tasks must be coordinated and therefore provide an interesting coupled configuration space to integrate mechanics, sensing, control, computation, and communication. Our work, over the past 20 years, has explicitly measured and quantified experimental trade-offs between

these resources (clock, planning/computation, synchronization, mechanics, sensing, communication) and also in removing or minimizing these resources. This has resulted in a series of novel devices, based on MEMS, for distributed manipulation surfaces, which represent design points with minimal resource profiles. A major challenge is the integration of mechanics, planning, manipulation, and control into the (currently) sensori-computational framework of information invariants. In short, information invariants can be seen as a theory of robot complexity when the robots are essentially mobile sensor networks. This results in a series of challenging and thought-provoking results, namely trade-offs in resources, and the ability to engineer systems that accomplish sophisticated tasks with surprisingly low resource-complexity in their design. A specific example, where we removed explicit communication and, instead, harness the ability of (multiple) robots to communicate through the task, is discussed in [23,48], for one application (moving large objects such as furniture). A key issue was removing synchronization to obtain an asynchronous distributed protocol (analogous to transformation (II*), above). The intellectual roots of this work spring from Dexter's 1978 paper, where he showed the equivalence of communication, internal state, and external state for maze-searching automata.

Acknowledgments. I would like to thank referees for their helpful suggestions on this article.

References

1. Ben-Or, M., Kozen, D., Reif, J.: The complexity of elementary algebra and geometry. Journal of Computer and System Sciences 32(2), 251–264 (1986)
2. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: Proceedings of the 19th Annual Symposium on Foundations of Computer Science, pp. 132–142. IEEE Computer Society (1978)
3. Blum, M., Sakoda, W.J.: On the capability of finite automata in 2 and 3 dimensional space. In: 18th Annual Symposium on Foundations of Computer Science, pp. 147–161 (1977)
4. Böhringer, K.-F., Bhatt, V., Donald, B.R., Goldberg, K.: Sensorless manipulation algorithms using a vibrating surface. Algorithmica 26(3/4), 389–429 (2000)
5. Böhringer, K.-F., Donald, B.R.: Algorithmic MEMS. In: Proceedings of the 3rd International Workshop on the Algorithmic Foundations of Robotics WAFR, Houston, TX (March 1998)
6. Böhringer, K.-F., Donald, B.R., Kovacs, G., MacDonald, N., Suh, J.: Computational methods for the design and control of MEMS micromanipulator arrays. IEEE Computational Science and Engineering 4(1), 17–29 (1997); Special Issue on Computational MEMS
7. Böhringer, K.-F., Donald, B.R., Lamiraux, F., Kavraki, L.: Part orientation with one or two stable equilibria using programmable force fields. IEEE Transactions on Robotics and Automation 16(2), 157–170 (2000)
8. Böhringer, K.-F., Donald, B.R., MacDonald, N.: Upper and lower bounds for programmable vector fields with applications to MEMS and vibratory plate parts feeders. In: Laumond, J.P., Overmars, M. (eds.) Algorithms for Robotic Motion and Manipulation, pp. 255–276. A. K. Peters, Wellesley (1997)

9. Böhringer, K.-F., Donald, B.R., MacDonald, N.C.: Programmable Vector Fields for Distributed Manipulation, with Applications to MEMS Actuator Arrays and Vibratory Parts Feeders. International Journal of Robotics Research 18(2) (February 1999)
10. Böhringer, K.-F., Donald, B.R., MacDonald, N.C., Mihailovich, R.: Sensorless manipulation using massively parallel micro-fabricated actuator arrays. In: Proc. IEEE International Conference on Robotics and Automation, San Diego, CA (May 1994)
11. Böhringer, K.-F., Donald, B.R., MacDonald, N.C., Mihailovich, R.: A theory of manipulation and control for microfabricated actuator arrays. In: Proc. 7th IEEE Workshop on Micro Electro Mechanical Systems (MEMS 1994), Kanagawa, Japan (January 1994)
12. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Undergraduate Texts in Mathematics. Springer, New York (1991)
13. Donald, B.R.: Error Detection and Recovery in Robotics. LNCS, vol. 336. Springer, Heidelberg (1989)
14. Donald, B.R.: Information invariants in robotics. Artificial Intelligence 72, 217–304 (1995)
15. Donald, B.R., et al.: MEMS Videos. Department of Computer Science, Duke University (July 2008), `http://www.cs.duke.edu/donaldlab/research_movies_mems.php`
16. Donald, B.R., et al.: Robotics Videos. Department of Computer Science, Duke University (July 2008), `http://www.cs.duke.edu/donaldlab/research_movies_robots.php`
17. Donald, B.R., Jennings, J., Rus, D.: Towards a theory of information invariants for cooperating autonomous mobile robots. In: Proceedings of the International Symposium of Robotics Research ISRR, Hidden Valley, PA (October 1993)
18. Donald, B.R., Kapur, D., Mundy, J.: Symbolic and Numerical Computation for Artificial Intelligence. Academic Press, Harcourt Jovanovich (1992)
19. Donald, B.R., Levey, C., McGray, C., Paprotny, I., Rus, D.: An untethered, electrostatic, globally-controllable MEMS micro-robot. Journal of Microelectromechanical Systems 15(1), 1–15 (2006)
20. Donald, B.R., Levey, C., Paprotny, I.: Planar microassembly by parallel actuation of MEMS microrobots. Journal of Microelectromechanical Systems 17(4), 789–808 (2008)
21. Donald, B.R., Xavier, P.: Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. Algorithmica 14(6), 443–479 (1995)
22. Donald, B.R., Xavier, P.: Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. Algorithmica 14(6), 480–530 (1995)
23. Donald, B.R., Jennings, J., Rus, D.: Information invariants for distributed manipulation. International Journal of Robotics Research 16(5), 673–702 (1997)
24. Donald, B.R., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. Journal of the ACM 40(5), 1048–1066 (1993)
25. Erdmann, M.: Using backprojections for fine motion planning with uncertainty. Int. J. Rob. Res. 5, 19–45 (1986)
26. Erdmann, M.: On Probabilistic Strategies for Robot Tasks. PhD thesis, MIT Department of EECS, MIT Department of EECS, MIT A.I. Lab, Cambridge MIT-AI-TR 1155 (1989)

27. Erdmann, M.: Towards task-level planning: Action-based sensor design. Technical report, Carnegie-Mellon, Carnegie-Mellon School of Computer Science, Tech. report, CMU-CS-92-116 (February 1991)
28. Erdmann, M., Mason, M.: An exploration of sensorless manipulation. In: Proceedings of the 1986 IEEE International Conference on Robotics and Automation, vol. 3, pp. 1569–1574 (April 1986)
29. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. J. Distrib. Comput. 1, 26–39 (1986)
30. Fisher, P.C.: Turing machines with restricted memory access. Information and Control 9(4), 364–379 (1966)
31. Yu Grigor'ev, D.: Complexity of deciding Tarski algebra. Journal of Symbolic Computation 5(1-2), 65–108 (1988)
32. Hopcroft, J.E., Schwartz, J.T., Sharir, M.: On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's problem". The International Journal of Robotics Research 3(3), 76–88 (1984)
33. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
34. Jennings, J., Donald, B.R.: Sensor interpretation and task-directed planning using perceptual equivalence classes. In: Proc. IEEE International Conference on Robotics and Automation, Sacramento, CA, pp. 190–197 (April 1991)
35. Jennings, J., Donald, B.R.: Constructive recognizability for task-directed robot programming. In: Proc. IEEE International Conference on Robotics and Automation, Nice, France, pp. 2446–2452 (May 1992)
36. Jennings, J., Donald, B.R.: Constructive recognizability for task-directed robot programming. Jour. Robotics and Autonomous Systems 9(1), 41–74 (1992) (invited)
37. Jennings, J., Rus, D.: Active model acquisition for near-sensorless manipulation with mobile robots. In: International Association of Science and Technology for Development (IASTED) International Conference on Robotics and Manufacturing, Oxford, England (1993)
38. Kozen, D.: Automata and planar graphs. In: Proceedings of the 2nd Symposium on Fundamentals of Computing Theory, FCT 1979, Berlin, pp. 243–254 (1979)
39. Lozano-Perez, T.: Spatial planning: A configuration space approach. IEEE Transactions on Computers C-32(2), 108–120 (1983)
40. Lozano-Pérez, T., Mason, M.T., Taylor, R.H.: Automatic synthesis of fine-motion strategies for robots. Int. J. of Robotics Research 3, 3–24 (1984)
41. Lumelsky, V.J., Stepanov, A.A.: Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 403–430 (1987)
42. Mason, M.T.: Mechanics and planning of manipulator pushing operations. The International Journal of Robotics Research 5(3), 53–71 (1986)
43. Minsky, M.L.: Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. The Annals of Mathematics 74(3), 437–455 (1961)
44. Natarajan, B.K.: On planning assemblies. In: Proceedings of the fourth Annual Symposium on Computational Geometry, pp. 299–308. ACM (1988)
45. Rees, J., Donald, B.R.: Program mobile robots in scheme. In: Proc. IEEE International Conference on Robotics and Automation, Nice, France, pp. 2681–2688 (May 1992)

46. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 421–427 (1979); Schwartz, J., Hopcroft, J., Sharir, M.: Planning, Geometry and Complexity of Robot Motion, ch. 11, pp. 267–281. Ablex publishing corp., New Jersey (1987)
47. Rosenschein, S.J.: Synthesizing information-tracking automata from environment descriptions. Technical report, Teleos Research TR No. 2 (1989)
48. Rus, D., Donald, B.R., Jennings, J.: Moving furniture with teams of autonomous mobile robots. In: Proc. IEEE/Robotics Society of Japan International Workshop on Intelligent Robots and Systems, Pittsburgh, PA, pp. 235–242 (1995)
49. Suh, J., Darling, R.B., Böhringer, K.-F., Donald, B.R., Baltes, H., Kovacs, G.: CMOS integrated organic ciliary actuator arrays for general-purpose micromanipulation tasks. Journal of Microelectromechanical Systems 8(4), 483–496 (1999)
50. Tarski, A.: A decision method for elementary algebra and geometry. Rand report. Rand Corp. (1948)

Subtyping for F-Bounded Quantifiers and Equirecursive Types

Neal Glew

Intel Labs, Santa Clara, CA
aglew@acm.org

1 Introduction

Equirecursive types consider a recursive type to be equal to its unrolling and have no explicit term-level coercions to change a term's type from the former to the latter or vice versa. This equality makes deciding type equality and subtyping more difficult than the other approach—*isorecursive types*, in which the types are not equal, but isomorphic, witnessed by explicit term-level coercions. Previous work has built intuition, rules, and polynomial-time decision procedures for equirecursive types for first-order type systems. Some work has been done for type systems with parametric polymorphism, but that work is incomplete (see below). This chapter will give an intuitive theory of equirecursive types for second-order type systems, sound and complete rules, and a decision procedure for subtyping.

Another interesting feature of type systems turns out to be quite related to equirecursive types. Canning et al. [CCH+89] introduced the idea of *F-bounded polymorphism*. In this form of polymorphism a type bound can mention the type being bounded. For example, it can require a type that has a method that returns an object of the type being bounded. This form of bound is useful for binary methods and in typing object encodings [Gle00]. Treating a type variable as being a subtype of its bound when that bound can refer to it is like treating a recursive type as being equal to its unrolling, and similar issues arise to formalising such type systems. This chapter will also treat F-bounded parameteric polymorphism, and give it an intuitive formalisation, sound and complete set of rules, and decision procedure for subtyping.

Amadio and Cardelli [AC93] were the first to investigate the equirecursive approach. They proposed the *tree interpretation* of recursive types, which is based on the idea of repeatedly unrolling recursive types into possibly-infinite trees. Two types are equal if their corresponding trees are the same; similarly, subtyping can be defined on trees and lifted to types. Amadio and Cardelli made these ideas precise, defined a set of rules for type equality and subtyping that are sound and complete, and provided an exponential-time decision procedure for equality and subtyping. Kozen et al. [KPS95] reduced this exponential time to quadratic time, by defining a notion of tree automata that generate trees just as types do and a construction from two tree automata that decides equality and subtyping. Both Amadio and Cardelli and Kozen et al. worked with first-order type systems. Colazzo and Ghelli [CG99] investigated a second-order type

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 66–82, 2012.

system with equirecursion. They gave a coinductive set of rules for subtyping and a decision procedure, but did not relate their rules to trees nor show soundness and completeness to any intuitive model. In previous work [Gle02a, Gle02b], I gave a tree model for second-order systems, a set of rules for equality of types, showed soundness and completeness of those rules, defined automata for the trees, and defined a construction on automata that gives a decision procedure for equality.[1] However, I did not address subtyping. Gauthier and Pottier [GP04] devised an $O(n \log n)$ algorithm to decide equality of second-order types with equirecursive types by reducing second-order types to canonical first-order types in a particular way such that the canonical types are equal exactly when the original types are equal; their approach also handles entailment of type equations and type unification problems with the same complexity.

This chapter investigates a second-order type system with F-bounded quantifiers and equirecursive types. First, it defines a notion of trees that provide an intuitive model for such types and defines an intuitive notion of subtyping on these trees. Next, it presents the types themselves and defines how these map to trees. Then it presents a set of type equality and subtyping rules and shows soundness and completeness of these rules with respect to the tree interpretation. Finally, it defines a notion of tree automata, how these generate trees, and a construction that decides subtyping in polynomial time.

Complete definitions and proofs of all the results in this chapter appear in a companion technical report [Gle12].

2 Binding Trees

I consider a system with top, bottom, function, and F-bounded forall quantified types. To model such types, I use possibly-infinite trees over these constructs and de Bruijn indices to model type variables. These trees are formulated in the standard way:

$$\begin{aligned} \mathit{Tree} = \{t : \{\mathsf{L},\mathsf{R}\}^* \rightharpoonup \mathbb{N} \cup \{\top, \bot, \to, \forall\} \mid \\ \epsilon \in \mathrm{dom}(t) \wedge (p\ell \in \mathrm{dom}(t) \Leftrightarrow t(p) \in \{\to, \forall\})\} \end{aligned}$$

Forall quantifiers are F-bounded, that means that $\forall$ binds a variable in each of its subtrees—in the left subtree, the bound, to refer to the type being bounded and in the right tree to refer to the quantified type.

Trees form a complete 1-bounded ultrametic space in the usual way. Some sample trees appear in Figure 1, and I overload the notation and use $\mathrm{var}(n)$, $\top$, $\bot$, $t_\mathsf{L} \to t_\mathsf{R}$, and $\forall t_\mathsf{L}.t_\mathsf{R}$ to denote appropriate trees. The subtree of t along path p is $\mathit{subtree}(t, p)$, the number of variables bound along the path from p_1 to p_2 (a suffix) is $\mathit{bind}(t, p_1 \to p_2)$, and shifting the free variables of a tree t up by n is $\mathit{shift}(t, n)$; their definitions are straightforward.

[1] In those papers I claimed that the algorithm could be made quadratic time. I have recently realised that the reasoning was incorrect, and thus that my papers demonstrated only an exponential-time algorithm. The arguments I give latter can be used to make an $O(n^4)$ algorithm from the ones in my previous work.

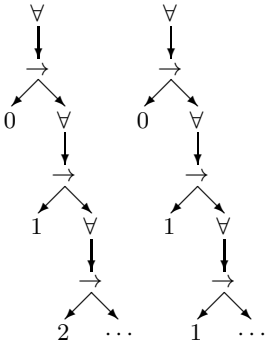


Fig. 1. Examples trees `t1`, on left, and `t2`, on right

2.1 Regular Trees

Not all trees are generated by syntactic types. For first-order systems, regular trees—those with a finite number of subtrees—correspond exactly to the trees that can be generated. For the trees defined above, this is no longer the case. Consider tree `t1` in Figure 1 that is generated by $\forall\alpha.\mathsf{rec}\ \beta.\alpha \to \forall\beta'.\alpha$—all the de Bruijn indices intuitively represent the same thing, the variable bound by the $\forall$ at the top of the tree, but the actual numbers are all different. Now consider tree `t2` in Figure 1. It has only a finite number of subtrees, but in fact no type can generate it. It looks like the second and subsequent forall trees repeat each other, however the de Bruijn indices 1 in that tree refer to the quantifier in the previous iteration of the cycle rather than the current iteration, and types cannot generate such a structure. My previous work defined a rather complicated notion of regular binding trees. Since then I have discovered a better, more intuitive formulation.

The idea is that relevant subtrees are equal modulo appropriate changes to the de Bruijn indices. For example, in the tree for $\mathsf{rec}\ \alpha.\forall\beta \leq \top.\alpha$, the tree itself is not equal to the right subtree, but the tree itself shifted by one is equal to the right subtree—one being the number of variables bound from the root to the right subtree. Taking the symmetric, transitive, and suffix closure of this idea provides a suitable notion of when two subtrees of a tree should be considered equivalent. Then a tree is regular if there are only a finite number of equivalent subtrees.

This idea can be formalised by defining an equivalence relation on the paths of a tree t, $eqst(t)$, that intuitively says which subtrees of t are equal. First, a tree being equivalent to one of its subtrees is formalised as:

$$\begin{aligned} eqst_t(p_1 \to p_2) = \ & \wedge\, p_2 \in \mathrm{dom}(t) \\ & \wedge\, \mathit{shift}(\mathit{subtree}(t, p_1), \mathit{bind}(t, p_1 \to p_2)) = \mathit{subtree}(t, p_2) \end{aligned}$$

Then the equivalence of paths of a tree is defined as:

$$\begin{aligned} eqst_t(R) = &\cup\{(p_1,p_2) \mid (p_2,p_1) \in R\} \\ &\cup\{(p_1,p_3) \mid (p_1,p_2) \in R \wedge (p_2,p_3) \in R\} \\ &\cup\{(p_1p,p_2p) \mid (p_1,p_2) \in R \wedge p_1p \in \mathrm{dom}(t) \wedge p_2p \in \mathrm{dom}(t)\} \\ &\cup\{(p_1,p_1p_2) \mid eqst_t(p_1 \to p_1p_2)\} \\ eqst(t) = &\cap_{eqst_t(R) \subseteq R} R \end{aligned}$$

It is easy to prove that $eqst(t)$ is an equivalence relation on $\mathrm{dom}(t)$. A tree t is a *regular binding tree* exactly when $[eqst(t)]$ is finite.

Each equivalence classes of $eqst(t)$ has a particular type constructor, as defined by the function:

$$NL(t,[p]_{eqst(t)}) = \begin{cases} \mathsf{fv}(n) & t(p) = n + bind(t, \epsilon \to p) \\ \mathsf{bv}([p']_{eqst(t)}, \ell) & p'\ell \le p \wedge t(p'_1) = \forall \wedge t(p) = bind(t, p'\ell \to p) \\ \top & t(p) = \top \\ \bot & t(p) = \bot \\ \to & t(p) = \to \\ \forall & t(p) = \forall \end{cases}$$

It is easy to prove that this function is well defined and that $[p]_{eqst(t)}\ell = [p\ell]_{eqst(t)}$ when $NL(t,[p]_{eqst(t)}) \in \{\to,\forall\}$ is also well defined.

2.2 Subtyping

Intuitively, subtyping of trees should be as follows. Top should be a supertype of everything, bottom should be a subtype of everything, a variable should be a subtype of itself and its bound, a function tree is a subtype of another function tree when the argument trees are related contravariantly and the result trees are related covariantly, and similarly for forall trees. This could be made into a formal definition by using coinduction except for a couple of points. First, if the coinductive definition includes the condition that a variable is a subtype of any tree its bound is a subtype of then problems result. In particular, if free variable 0 is bounded by itself then under this definition free variable 0 is a subtype of any tree because of the coinduction—what we really want is induction for bounds. Second, comparing $\forall t_{1\mathsf{L}}.t_{1\mathsf{R}}$ to $\forall t_{2\mathsf{L}}.t_{2\mathsf{R}}$ requires selecting a bound for the variable bound by the $\forall$s to do the comparison of $t_{1\mathsf{L}}$ against $t_{2\mathsf{L}}$ and $t_{1\mathsf{R}}$ against $t_{2\mathsf{R}}$. The most general rule that is sound uses the tightest bound for the variable—this is $t_{2\mathsf{L}}$. However, that rule leads to an undecidable subtyping relation [Pie94]. The system considered here is actually a non-conversative extension,[2] but I believe that the undecidability still holds (but have not proven this yet). Therefore, to regain decidability, I will use the Kernel rule for forall—the bound is required to be invariant ($t_{1\mathsf{L}} = t_{2\mathsf{L}}$) and then the bound to use is the equal bound.

[2] The system considered by Pierce does not include recursive types of either flavour, but the rules in this paper could be used to define a more permissive subtyping relation that is still sound for Pierce's system—in essence certain subtyping that has an infinite derivation in Pierce's rules would be allowed rather than rejected.

To formalise the above intuition, there are several pieces to build up, as mixing induction and coinduction is a little tricky. A bound set, ranged over by metavariable β, is a function from de Bruijn indices to trees, $BSet = \mathbb{N} \to Tree$. Shifting the free variables up and adding a bound t for the new free variable 0 is $shift(\beta, t)$, and is straightforward to define. Tree promotion, a relation $\hookrightarrow$ on $Tree \times BSet \times Tree$, is defined as $\text{var}(n) \hookrightarrow_\beta \beta(n)$ (no other trees are related by $\hookrightarrow_\beta$). The base subtyping proposition $bst(t_1, R, \beta, t_2)$ holds exactly when either:

- $t_1 = \text{var}(n)$ and $t_2 = \text{var}(n)$,
- $t_2 = \top$,
- $t_1 = \bot$,
- $t_1(\epsilon) = \to$, $t_2(\epsilon) = \to$, $subtree(t_2, \mathsf{L})\ R_\beta\ subtree(t_1, \mathsf{L})$, and $subtree(t_1, \mathsf{R})\ R_\beta\ subtree(t_2, \mathsf{R})$, or
- $t_1(\epsilon) = \forall$, $t_2(\epsilon) = \forall$, $subtree(t_1, \mathsf{L}) = subtree(t_2, \mathsf{L})$, and:

$$subtree(t_1, \mathsf{R})\ R_{shift(\beta, subtree(t_2, \mathsf{L}))}\ subtree(t_2, \mathsf{R})$$

A three place relation R on $Tree \times BSet \times Tree$ is a partial subtyping exactly when $t_1\ R_\beta\ t_2$ implies that there exists t_1' such that $t_1 \hookrightarrow_\beta^* t_1'$ and $bst(t_1', R, \beta, t_2)$. Subtyping for trees, $\leq$, is the union of all partial subtypings.

Subtyping satisfies several important properties justifying that the formal definitions do capture the right intuition.

Theorem 1. *Subtyping is a partial subtyping; $\leq_\beta$ is a preorder on Tree for any $\beta \in BSet$; $t_1 \leq_\beta t_2$ if and only if one of the following holds:*

- $t_1 = \text{var}(n)$ *and* $t_2 = \text{var}(n)$,
- $t_1 = \text{var}(n)$ *and* $\beta(n) \leq_\beta t_2$,
- $t_2 = \top$,
- $t_1 = \bot$,
- $t_1 = t_{11} \to t_{12}$, $t_2 = t_{21} \to t_{22}$, $t_{21} \leq_\beta t_{11}$, *and* $t_{12} \leq_\beta t_{22}$, *or*
- $t_1 = \forall t_3.t_4$, $t_2 = \forall t_3.t_5$, *and* $t_4 \leq_{shift(\beta, t_3)} t_5$.

2.3 Characterising Subtyping

A (tree) subtyping problem is a triple $(t_\mathsf{L}, \beta, t_\mathsf{R})$ where $t_\mathsf{L} \leq_\beta t_\mathsf{R}$ might or might not hold. The definition of subtyping says that after promoting the subtype to its bound some finite number of times the two trees have to match in a certain sense. In particular, two trees match when they are the same de Bruijn index, the supertype is top, the subtype is bottom, both trees are functions, or both trees are forall quantifiers and their respective left subtrees are equal. With this definition, subtyping requires that after a finite number of promotions of the subtype the two trees must match and furthermore, if they match because they are functions then the respective left subtrees must be contravariantly related and the respective right subtrees must be covariantly related, and if they match because they are forall quantifiers then the respective right subtrees must be covariantly related. For these various subtrees we can repeat this process, finding

matching trees for them, and so on. Thus for some set of paths we get trees that match if the original subtyping held.

We can formalise this idea as follows. A subproblem of *stp* will be a triple $(t_{\mathsf{L}}, \beta, t_{\mathsf{R}})$ where t_{L} is the current subtype, t_{R} is the current supertype, and β is the current bound set. For a subtyping problem *stp*, the initial subproblem for path ϵ is simply *stp*. If the initial subproblem for path p is $(t_{\mathsf{L}}, \beta, t_{\mathsf{R}})$ and there is a t'_{L} such that $t_{\mathsf{L}} \hookrightarrow^*_\beta t'_{\mathsf{L}}$ and t'_{L} and t_{R} match then the final subproblem for path p is $(t'_{\mathsf{L}}, \beta, t_{\mathsf{R}})$; otherwise the final subproblem for path p fails. If the final subproblem for path p is $(t_{\mathsf{L}}, \beta, t_{\mathsf{R}})$ and matches because both trees are functions then the initial subproblem for path $p\mathsf{L}$ is $(\mathit{subtree}(t_{\mathsf{R}}, \mathsf{L}), \beta, \mathit{subtree}(t_{\mathsf{L}}, \mathsf{L}))$ and the initial subproblem for path $p\mathsf{R}$ is $(\mathit{subtree}(t_{\mathsf{L}}, \mathsf{R}), \beta, \mathit{subtree}(t_{\mathsf{R}}, \mathsf{R}))$. Similarly, if both trees are forall quantifiers then the initial subproblem for path $p\mathsf{R}$ is $(\mathit{subtree}(t_{\mathsf{L}}, \mathsf{R}), \mathit{shift}(\beta, \mathit{subtree}(t_{\mathsf{L}}, \mathsf{L})), \mathit{subtree}(t_{\mathsf{R}}, \mathsf{R}))$. Thus any subtyping problem has a prefixed closed set of paths containing ϵ of initial and final subproblems.

The characterisation of subtyping is the following theorem.

Theorem 2. $t_{\mathsf{L}} \leq_\beta t_{\mathsf{R}}$ *if and only if all final subproblems of* $(t_{\mathsf{L}}, \beta, t_{\mathsf{R}})$ *do not fail.*

We can go further than just these definitions however. Each tree that appears in a subproblem comes, in some sense, from either of the original trees or one of the bounds. De Bruijn indices that are bounded by themselves (a trivial bound) are not interesting, so a tree identifier for a subtyping problem $(t_{\mathsf{L}}, \beta, t_{\mathsf{R}})$ is either L, R, or n where $\beta(n) \neq \mathrm{var}(n)$. A node identifier is a tree identifier and a sequence of L, R, and Ss specifying to take the left subtree, right subtree, or shift by one starting from that tree. Any subproblem of *stp* can be represented as a triple $(ni_{\mathsf{L}}, nis, ni_{\mathsf{R}})$ where ni_{L} is a node identifier representing the subtype, ni_{R} is a node identifiers representing the supertype, and nis is a sequence of node identifiers representing the bounds of the binding variables that have been opened up. We can inductively define two partial maps that compute the node identifier representations of the initial and final subproblems or F for a final subproblem that fails.

Each node identifier maps to an equivalence class of the equivalence of subtrees of the tree that it comes from. If there are only finitely many de Bruijn indices with non-trivial bounds, each of those is a regular binding tree, and the original subtype and supertype trees are regular binding trees, then the node identifiers map to a finite set of equivalence classes. So the pair of the current subtype and supertype node identifiers map to a finite set. Thus for any sufficiently long path for which there are initial or final subproblems there will be a repeat of this pair. This property is the key to showing completeness of the subtyping rules, it will be used to cut off the proof of subtyping to make a finite derivation, as we shall see.

This characterisation of subtyping is also used to build an algorithm for deciding subtyping. Essentially, the algorithm constructs a deterministic finite-state automata that searches for paths for which the final subproblem fails. If it fails

to find such a path, that is, its language is empty, then the subtyping holds. It uses just the equivalence classes of the node identifiers as well as some information about which binder equivalence classes correspond—but this information is finite too if there are finitely many equivalence classes. The algorithm is able to do promotion, matching, and determining subtrees with just this information. If there are a finite number of equivalence classes (as will be the case for regular binding trees and finite number of non-trivial bounds) then the search space is finite and the algorithm is a decision procedure.

3 Types

Now I will define the syntactic type system, map it to trees, and then give a sound and complete set of type equality and subtyping rules.

Let *Var* be a some set of type variables ranged over by metavariable α. The set of types, *Type*, ranged over by metavariables τ and σ is defined by this grammar:

$$\tau ::= \alpha \mid \top \mid \bot \mid \tau_1 \to \tau_2 \mid \forall \alpha \leq \tau_1 . \tau_2 \mid \mathsf{rec}\ \alpha . \tau$$

subject to the requirement that in $\mathsf{rec}\ \alpha.\tau$, τ is syntactically contractive in α, written $\tau \downarrow \alpha$. The latter is defined by induction on the structure of τ as follows:

$$\begin{array}{ll}
\alpha' \downarrow \alpha & \Leftarrow \alpha \neq \alpha' \\
\top \downarrow \alpha & \\
\bot \downarrow \alpha & \\
\tau_1 \to \tau_2 \downarrow \alpha & \\
\forall \alpha' \leq \tau_1 . \tau_2 \downarrow \alpha & \\
\mathsf{rec}\ \alpha' . \tau \downarrow \alpha & \Leftarrow \alpha = \alpha' \vee \tau \downarrow \alpha
\end{array}$$

3.1 Mapping Types to Regular Binding Trees

Types map to trees given trees for the free variables. An environment, ranged over by metavariable η, maps type variables to trees, $Env = Var \to Tree$. An environment is distinguishing if it maps type variables injectively to $\{\mathrm{var}(n) \mid n \in \mathbb{N}\}$ (note this is weaker than my previous work, which required bijectivity). Shifting the free variables of an environment η up and mapping α to the new free variable 0 is $shift(\eta, \alpha)$ and is straightforward to define.

The meaning of a type in an environment is a tree defined by induction on the type as follows:

$$\begin{array}{ll}
treeof(\alpha)_\eta & = \eta(\alpha) \\
treeof(\top)_\eta & = \top \\
treeof(\bot)_\eta & = \bot \\
treeof(\tau_1 \to \tau_2)_\eta & = treeof(\tau_1)_\eta \to treeof(\tau_2)_\eta \\
treeof(\forall \alpha \leq \tau_1 . \tau_2)_\eta & = \forall treeof(\tau_1)_{shift(\eta,\alpha)} . treeof(\tau_2)_{shift(\eta,\alpha)} \\
treeof(\mathsf{rec}\ \alpha . \tau)_\eta & = \mathrm{fix}(\lambda t . treeof(\tau)_{\eta\{\alpha \mapsto t\}})
\end{array}$$

where $\mathrm{fix}(f)$ is the unique fixed point of a contractive function f on trees (complete ultrametic spaces have unique fixed points for contractive functions)—the definition here is well defined as it is easy to prove that syntactic contractivity implies contractivity.

The meaning of a type is a regular binding tree and any regular binding tree is the meaning of some type.

Theorem 3. *If η maps type variables to regular binding trees then $\mathit{treeof}(\tau)_\eta$ is a regular binding tree. If t is a regular binding tree and η can generate t's free de Bruijn indices (there exists an α such that $\eta(\alpha) = \mathrm{var}(n)$ for each n such that $t(p) = n + \mathit{bind}(t, \epsilon \to p)$) then there is a type τ such that $\mathit{treeof}(\tau)_\eta = t$.*

3.2 Equality and Subtyping Rules

To motivate the subtyping rules, consider some particular problems. First, if α is bounded by $\top{\to}\alpha$ then should α be a subtype of $\mathsf{rec}\ \alpha'.\top{\to}\alpha'$? Intuitively, α is some set A of functions that take any value to a value in A; similarly $\mathsf{rec}\ \alpha'.\top{\to}\alpha'$ is the set B of all functions that take any value to a value in B; it seems that A should be a subset of B, so the subtyping should hold. Using de Bruijn index 0 for α then these types map to the trees $\mathrm{var}(0)$ and $t_2 = \{(\mathsf{R}^*, \to), (\mathsf{R}^*\mathsf{L}, \top)\}$ with bound set β such that $\beta(0) = \top{\to}\mathrm{var}(0)$. Let $R = \{(\mathrm{var}(0), \beta, t_2), (\top, \beta, \top)\}$. Then R is a partial subtyping and so $\mathrm{var}(0) \leq_\beta t_2$. If the subtyping rules are to be complete then clearly they must be able to derive that α bounded by $\top{\to}\alpha$ is a subtype of $\mathsf{rec}\ \alpha'.\top{\to}\alpha'$.

Second, consider an example that does not even involve recursive types. If α is bounded by $(\alpha{\to}\top){\to}\bot$ then should α be a subtype of $\alpha{\to}\top$? These types map to trees $\mathrm{var}(0)$ and $\mathrm{var}(0){\to}\top$ with β such that $\beta(0) = (\mathrm{var}(0){\to}\top){\to}\bot$. Let $R = \{(\mathrm{var}(0), \beta, \mathrm{var}(0){\to}\top), (\bot, \beta, \top)\}$. Then R is a partial subtyping so $\mathrm{var}(0) \leq_\beta \mathrm{var}(0){\to}\top$ and the subtyping for the types above should be derivable with the subtyping rules.

Using the standard structural subtyping rules with the equality rules from Amadio and Cardelli to try to prove these subtypings results in a cycle—after some steps what needs to be proved is what we are trying to prove. Here is the attempt for the first subtyping (where $B = \alpha \leq \top{\to}\alpha$ and $\tau_2 = \mathsf{rec}\ \alpha'.\top{\to}\alpha'$):

$$\dfrac{\dfrac{B \vdash \alpha \leq \top{\to}\alpha}{B \vdash \alpha \leq \tau_2}\quad \dfrac{\dfrac{\dfrac{}{B \vdash \top \leq \top}\quad B \vdash \alpha \leq \tau_2}{B \vdash \top{\to}\alpha \leq \top{\to}\tau_2}\quad \dfrac{\dfrac{}{\vdash \tau_2 = \top{\to}\tau_2}}{\vdash \top{\to}\tau_2 = \tau_2}}{B \vdash \top{\to}\tau_2 \leq \tau_2}}{B \vdash \top{\to}\alpha \leq \tau_2}}$$

Notice though that the steps make some progress, in that they use the structural subtyping rule for function types at least once, so coinductive proofs would prove this subtyping. A specialised rule for recursive types could prove this derivation, but in the other example, we really need something like coinduction (where $B = \alpha \leq (\alpha{\to}\top){\to}\bot$):

$$\dfrac{\dfrac{}{B \vdash \alpha \leq (\alpha{\to}\top){\to}\bot} \quad \dfrac{B \vdash \alpha \leq \alpha{\to}\top \quad \overline{B \vdash \bot \leq \top}}{B \vdash (\alpha{\to}\top){\to}\bot \leq \alpha{\to}\top}}{B \vdash \alpha \leq \alpha{\to}\top}$$

I will present normal inductive rules that in the rules that make progress, namely the structural subtyping rules for function and forall quantified types, allow the conclusion to be assumed in proving the subterms to have the appropriate subtyping relationship. This modification of the standard rules is enough to get sound and complete rules with respect to the tree interpretation of types.

Subtyping assumptions, ranged over by metavariable A, are sets of pairs of types, which I will write in the form $\tau_1 \leq \sigma_1, \ldots, \tau_n \leq \sigma_n$. Subtyping bounds, ranged over by metavariable B, have the form $\alpha_1 \leq \tau_1, \ldots, \alpha_n \leq \tau_n$ where the α_i are distinct. The meaning of subtyping bounds in a distinguishing environment is a bound set and is defined as:

$$\mathit{treeof}(\alpha_1 \leq \tau_1, \ldots, \alpha_n \leq \tau_n)_\eta = \lambda n. \begin{cases} \mathit{treeof}(\tau_i)_\eta & \eta(\alpha_i) = \mathrm{var}(n) \\ \mathrm{var}(n) & \text{otherwise} \end{cases}$$

The rules for type equality and subtyping appear in Figure 2 and define two judgements. $\vdash \tau_1 = \tau_2$ asserts that types τ_1 and τ_2 are equal and $A; B \vdash \tau_1 \leq \tau_2$ asserts that τ_1 is a subtype of τ_2 under assumptions A and bounds B. The equality rules are those of Amadio and Cardelli. The interesting rules are EQROLL and EQUNQ. The former asserts that a recursive type is equal to its unrolling. The latter asserts that recursive types are unique—more specifically that two types that satisfy the same syntactically contractive equation are equal. It is key to proving completeness of the equality rules with respect to the tree interpretation of types. The subtyping rules are also fairly standard. There are the usual reflexivity, transitivity, variable bound, top, and bottom rules. Rule STASSUME allows an assumption to be used. Rule STFUN is the usual structural subtyping rule except that the conclusion can be assumed while proving the the argument types are contravariantly related and the result types are covariantly related. Rule STALL is the Kernel rule for F-bounded forall quantified types, again where the conclusion can be assumed when proving that the body types are covariantly related.

We are mainly interested in judgements of the form $\emptyset; B \vdash \tau_1 \leq \tau_2$, which I will write simply as $B \vdash \tau_1 \leq \tau_2$—the assumption sets are really just for internal use to prove such judgements.

I proved the soundness and completeness of the equality rules in previous work [Gle02a]. I repeat those proofs for the system in this chapter in the companion technical report [Gle12], and I will use them in proving the soundness and completeness of the subtyping rules.

3.3 Soundness

If two types are subtypes then the trees that they map to are subtypes.

$\boxed{\vdash \tau_1 = \tau_2}$

$$\frac{\vdash \tau_2 = \tau_1}{\vdash \tau_1 = \tau_2}\ \text{EQSYM} \qquad \frac{\vdash \tau_1 = \tau_2 \quad \vdash \tau_2 = \tau_3}{\vdash \tau_1 = \tau_3}\ \text{EQTRANS}$$

$$\frac{}{\vdash \alpha = \alpha}\ \text{EQVAR} \qquad \frac{}{\vdash \top = \top}\ \text{EQTOP} \qquad \frac{}{\vdash \bot = \bot}\ \text{EQBOT}$$

$$\frac{\vdash \tau_1 = \tau_2 \quad \vdash \sigma_1 = \sigma_2}{\vdash \tau_1 \to \sigma_1 = \tau_2 \to \sigma_2}\ \text{EQFUN} \qquad \frac{\vdash \tau_1 = \tau_2 \quad \vdash \sigma_1 = \sigma_2}{\vdash \forall \alpha \leq \tau_1.\sigma_1 = \forall \alpha \leq \tau_2.\sigma_2}\ \text{EQALL}$$

$$\frac{\vdash \tau_1 = \tau_2}{\vdash \mathsf{rec}\ \alpha.\tau_1 = \mathsf{rec}\ \alpha.\tau_2}\ \text{EQREC} \qquad \frac{}{\vdash \mathsf{rec}\ \alpha.\tau = \tau\{\alpha \mapsto \mathsf{rec}\ \alpha.\tau\}}\ \text{EQROLL}$$

$$\frac{\vdash \tau_1 = \sigma\{\alpha \mapsto \tau_1\} \quad \vdash \tau_2 = \sigma\{\alpha \mapsto \tau_2\} \quad \sigma \downarrow \alpha}{\vdash \tau_1 = \tau_2}\ \text{EQUNQ}$$

$\boxed{A; B \vdash \tau_1 \leq \tau_2}$

$$\frac{\vdash \tau_1 = \tau_2}{A; B \vdash \tau_1 \leq \tau_2}\ \text{STREF} \qquad \frac{A; B \vdash \tau_1 \leq \tau_2 \quad A; B \vdash \tau_2 \leq \tau_3}{A; B \vdash \tau_1 \leq \tau_3}\ \text{STTRANS}$$

$$\frac{\tau_1 \leq \tau_2 \in A}{A; B \vdash \tau_1 \leq \tau_2}\ \text{STASSUME} \qquad \frac{\alpha \leq \tau \in B}{A; B \vdash \alpha \leq \tau}\ \text{STBOUND}$$

$$\frac{}{A; B \vdash \tau \leq \top}\ \text{STTOP} \qquad \frac{}{A; B \vdash \bot \leq \tau}\ \text{STBOT}$$

$$\frac{\begin{array}{c} A' = A, \tau_1 \to \sigma_1 \leq \tau_2 \to \sigma_2 \\ A'; B \vdash \tau_2 \leq \tau_1 \\ A'; B \vdash \sigma_1 \leq \sigma_2 \end{array}}{A; B \vdash \tau_1 \to \sigma_1 \leq \tau_2 \to \sigma_2}\ \text{STFUN} \qquad \frac{\begin{array}{c} A' = A, \forall \alpha \leq \tau_1.\sigma_1 \leq \forall \alpha \leq \tau_2.\sigma_2 \\ \vdash \tau_1 = \tau_2 \\ A'; B, \alpha \leq \tau_2 \vdash \sigma_1 \leq \sigma_2 \\ \alpha \notin fv(A) \cup fv(B) \end{array}}{A; B \vdash \forall \alpha \leq \tau_1.\sigma_1 \leq \forall \alpha \leq \tau_2.\sigma_2}\ \text{STALL}$$

Fig. 2. Typing Rules

Theorem 4. *If $B \vdash \tau_1 \leq \tau_2$ and η is distinguishing then $treeof(\tau_1)_\eta \leq_{treeof(B)_\eta} treeof(\tau_2)_\eta$.*

Proof: The proof uses a generalisation of subtyping on trees that takes assumptions into account, and is then by induction over the derivation of the subtyping judgement using a lemma that says that subtyping with assumptions has properties similar to those of the rules. The soundness of the equality rules is also used for Rule STREF. □

3.4 Completeness

If the trees two types map to are subtypes then the rules can derive that they are subtypes.

Theorem 5. *If η is distinguishing and $treeof(\tau_1)_\eta \leq_{treeof(B)_\eta} treeof(\tau_2)_\eta$ then $B \vdash \tau_1 \leq \tau_2$.*

The proof is in the companion technical report [Gle12]. As previously mentioned, the key to the proof is the characterisation of subtyping. It states that no final

subproblem fails. Each final subproblem can be represented using node identifiers and these node identifiers come from a finite set of equivalence classes, thus on any sufficiently long path there will be a repeat of which equivalence class is the subtype and which equivalence class is the supertype. For any node identifier of the initial and final subproblems the proof builds a canonical type. Then the proof shows that Rule STBOUND can mimic promotion to a bound and that if two node identifiers match then one of the Rules EQVAR and STREF, STTOP, STBOT, STFUN, or STALL can prove the required subtyping. For STALL the proof uses the fact that the left subtrees are equal and the completeness of the equality rules to show that the bound types are equal; for the right subtree and both subtrees of STFUN the proof recurses to a longer path, for which there are initial and final subproblems. Finally at a repeat in the equivalence classes the proof shows that the requried subtyping is in the assumption set and uses Rule STASSUME. Finally, in various places the proof needs to show that the canonical types match up to other types, which it does by showing that they generate the same trees and by using the completeness of the equality rules; Rule STTRANS is used to combine everything together. The proof is just going through all the details of the above sketch.

4 Binding-Tree Automata

This section defines a notion of tree automata that generate trees and a construction that determines subtyping—it takes two tree automata to a DFA whose language is empty exactly when the subtyping relation holds.

A binding-tree automata is a quadruple (Q, i, δ, lf) such that Q is a finite set of states, $i \in Q$ is the initial state, $\delta : Q \times \{\mathsf{L}, \mathsf{R}\} \rightharpoonup Q$ is the transition function, $lf : Q \to \{\mathsf{fv}(n) \mid n \in \mathbb{N}\} \cup \{\mathsf{bv}(q, \ell) \mid q \in Q \wedge \ell \in \{\mathsf{L}, \mathsf{R}\}\} \cup \{\top, \bot, \to, \forall\}$ is the labelling function, $\delta(q, \ell)$ is defined if and only if $lf(q) \in \{\to, \forall\}$, and $lf(q) = \mathsf{bv}(q', \ell)$ only if $lf(q') = \forall$ and all paths from i to q go through q' and on the last time through q' they follow an ℓ edge. Intuitively, a tree automata takes as input a path through a tree and outputs the node label at the end of that path, which can be either a free variable (of the original tree), a bound variable (that bound by the last time through the identified state), top, bottom, function, or forall.

The tree that a tree automata generates is defined as follows:

$$
\begin{aligned}
bind(ql, \ell) &= \begin{cases} 1 & ql = \forall \\ 0 & ql \neq \forall \end{cases} \\
shift(f, q := n) &= \lambda q'. \begin{cases} 0 & q' = q \\ f(q') + n & q' \neq q \end{cases} \\
\hat{\delta}((q, n, f), \ell) &= (\delta(q, \ell), n + bind(lf(q), \ell), shift(f, q := bind(lf(q), \ell))) \\
\hat{lf}(q, n, f) &= \begin{cases} n + m & lf(q) = \mathsf{fv}(m) \\ f(q') & lf(q) = \mathsf{bv}(q', \ell) \\ lf(q) & \text{otherwise} \end{cases} \\
treeof(Q, i, \delta, lf) &= \lambda p. \hat{lf}(\hat{\delta}^*((i, 0, \lambda q.0), p))
\end{aligned}
$$

where δ^* is the obvious lifting of δ to sequences of edges. The formal definition just tracks enough information to determine the de Bruijn indices for the states labelled as free and bound variables, otherwise it follows the intuition above.

Binding-tree automata generate regular binding trees and all regular binding trees are generated by a binding-tree automata.

Theorem 6. *If ta is a binding-tree automata then $\mathit{treeof}(ta)$ is a regular binding tree. If t is a regular binding tree then there exists a binding-tree automata ta such that $\mathit{treeof}(ta) = t$.*

4.1 Subtyping Algorithm

Now, I will define a construction that takes two binding-tree automata and produces a deterministic finite-state automata (in the usual sense), such that the DFA's language is empty if and only if the trees of the two binding-tree automata are in the subtyping relation. In particular, the DFA will search for paths that show that the two trees are not subtypes. The characterisation of subtyping tells us that such paths exists if and only if the trees are not subtypes. Most of the information for determining if states match after promotion is available from the labelling functions, but some additional information is needed. Specifically, the construction must track which binding states in one binding-tree automata correspond to which binding states in the other binding-tree automata, in order to determine if two bound variables match or not. This correspondence will be tracked by *partial bijections*, defined next.

A partial bijection R between sets A and B is a set of pairs from A and B such that $(a_1, b_1) \in R$ and $(a_2, b_2) \in R$ implies that $a_1 = a_1$ if and only if $b_1 = b_2$. Partial bijection update is defined as: $R\{a \leftrightharpoons b\} = \{(a', b') \in R \mid a' \neq a \wedge b' \neq b\} \cup \{(a, b)\}$.

An automata bounds is a finite function from de Bruijn indices to binding-tree automata—de Bruijn indices without a bound are bounded by themselves. An automata bounds generates a bound set as follows:

$$\mathit{treeof}(ba) = \lambda n. \begin{cases} \mathit{treeof}(ba(n)) & n \in \mathrm{dom}(ba) \\ \mathrm{var}(n) & n \notin \mathrm{dom}(ba) \end{cases}$$

The input to the construction, an (automata) subtyping problem, is a triple $(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}})$ where ta_{L} and ta_{R} are binding-tree automata and ba is an automata bounds. The construction will search over the various states of the various automata, so define a problem state of $(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}})$ to be either (L, q) for q a state of ta_{L}, (n, q) for $n \in \mathrm{dom}(ba)$ and q a state of $ba(n)$, or (R, q) for q a state of ta_{R}. Define the transition function, δ, and the labelling function, lf, for $(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}})$ by lifting the underlying transition functions and labelling functions in the obvious way. The states of the DFA are quadruples (q_1, ϕ, q_2, R) where q_1 and q_2 are problem states, $\phi \in \{+, \circ, -\}$ is a variance ($+$ means that q_1 should be a subtype of q_2; $\circ$ means that q_1 should be equal to q_2; $-$ means that q_1 should be a supertype of q_2), and R is a partial bijection between problem states.

I build the formal definition of the construction up in several pieces. First, I define how problem states are promoted, that is, if they are variables they are replaced with their bounds, as follows: $q \hookrightarrow_{(ta_{\mathsf{L}},ba,ta_{\mathsf{R}})} (n,i)$ if $lf(q) = \mathsf{fv}(n)$, $n \in \mathrm{dom}(ba)$, and i is the initial state of $ba(n)$; and $q \hookrightarrow_{stp} \delta(q', \mathsf{L})$ if $lf(q) = \mathsf{bv}(q', \ell)$.

Second, a DFA state matches, $matches_{stp}(q_1, \phi, q_2, R)$, exactly when one of the following holds:

- $lf(q_1) = lf(q_2) = \mathsf{fv}(n)$,
- $lf(q_1) = \mathsf{bv}(q_1', \ell)$, $lf(q_2) = \mathsf{bv}(q_2', \ell)$, and $(q_1', q_2') \in R$,
- $lf(q_1) = \top$ and $\phi = -$, $lf(q_2) = \top$ and $\phi = +$, or $lf(q_1) = lf(q_2) = \top$,
- $lf(q_1) = \bot$ and $\phi = +$, $lf(q_2) = \bot$ and $\phi = -$, or $lf(q_1) = lf(q_2) = \bot$,
- $lf(q_1) = lf(q_2) = \rightarrow$, or
- $lf(q_1) = lf(q_2) = \forall$.

Intuitively, a state matches if the base subtyping proposition holds for the nodes represented by that state. Using it and the notion of promotion, I can define a function that promotes a DFA state if possible to a matching DFA state. In particular, define $promote_{stp}(q_1, +, q_2, R) = (q_1', +, q_2, R)$ where q_1' is the first q_1' such that $q_1 \hookrightarrow^*_{stp} q_1'$ and $matches_{stp}(q_1', +, q_2, R)$ or $q_1' = q_1$ if no such q_1' exists; similarly, define $promote_{stp}(q_1, -, q_2, R) = (q_1, -, q_2', R)$ where q_2' is the first q_2' such that $q_2 \hookrightarrow^*_{stp} q_2'$ and $matches_{stp}(q_1, -, q_2', R)$ or $q_2' = q_2$ if no such q_2' exists; define $promote_{stp}(q_1, \circ, q_2, R) = (q_1, \circ, q_2, R)$.

Third, the subtree of a DFA state along an edge is defined as follows:

$$\begin{array}{ll}
subtree_{stp}((q_1, \phi, q_2, R), \mathsf{L}) = (\delta(q_1, \mathsf{L}), -\phi, \delta(q_2, \mathsf{L}), R\{q_1 \leftrightharpoons q_2\}) & lf(q_1) = \rightarrow \\
subtree_{stp}((q_1, \phi, q_2, R), \mathsf{L}) = (\delta(q_1, \mathsf{L}), \circ, \delta(q_2, \mathsf{L}), R\{q_1 \leftrightharpoons q_2\}) & lf(q_1) = \forall \\
subtree_{stp}((q_1, \phi, q_2, R), \mathsf{R}) = (\delta(q_1, \mathsf{R}), \phi, \delta(q_2, \mathsf{R}), R\{q_1 \leftrightharpoons q_2\}) & lf(q_1) \in \{\rightarrow, \forall\}
\end{array}$$

(Where $-+ = -$, $-\circ = \circ$, and $-- = +$.) Intuitively it computes the state that corresponds to the left or right subproblem of a DFA state, updating the variance and binding correspondence in the appropriate way.

Finally, the subtype automata is defined as follows:

$$\begin{array}{l}
subtype(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}}) = \\
\quad (Q \times Var \times Q \times (Q \leftrightharpoons Q), \\
\quad promote_{stp}(q_{\mathsf{L}}, +, q_{\mathsf{R}}, \emptyset), \\
\quad \lambda(q, \ell).promote_{stp}(subtree_{stp}(q, \ell)), \\
\quad \{q \mid \neg matches_{stp}(q)\})
\end{array}$$

where Q is the set of problem states of $(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}})$, q_{L} is the initial problem state of ta_{L}, and q_{R} is the initial problem state of ta_{R}.

The DFA so constructed has an empty language exactly when the subtyping relation holds.

Theorem 7. *$L(subtype(ta_{\mathsf{L}}, ba, ta_{\mathsf{R}})) = \emptyset$ if and only if $treeof(ta_{\mathsf{L}}) \leq_{treeof(ba)} treeof(ta_{\mathsf{R}})$.*

Proof: (Sketch) First the proof shows that for all the paths for which there are initial subproblems that the states computed by the subtype DFA (in some

sense) generate the corresponding trees of the subproblem. If the subtyping holds then all the final subproblems do not fail. The proof then shows that those paths match in the subtype DFA and are thus not in the language. The only other paths the DFA considers are following a left edge from a forall matching subproblem, where the DFA switches to invariance and becomes an equality checker. Since to forall match the left subtrees must be equal, none of those paths will be in the language. Conversely if the language is empty then the proof shows that the left edge from forall matching states implies the left subtrees are equal and thus the corresponding final subproblem does not fail, and that all other final subproblems do not fail from their paths not being in the language. Thus by characterisation of subtyping the subtyping holds. The rest of the proof is just working through all the tedious details. □

Since determining if the language of a DFA is linear time, the construction provides an exponential time algorithm for deciding subtyping (at least of automata).

4.2 Polynomial Time Algorithm

The key to getting a polynomial time algorithm is that the binder correspondence information is only used in very limited ways in the subtype DFA, and so it can almost be ignored. For this section, fix a subtyping problem *stp*. Let Q be the problem states of *stp*, *lf* the labelling function for problem states, and $\mathit{subtype}(\mathit{stp}) = (Q', i, \delta', F)$.

A triple is binder correspondence independent, $\mathit{bci}_{\mathit{stp}}(q_1, \phi, q_2)$, exactly when $\phi = +$ and $\mathit{lf}(q_2)$ is not a bound variable, $\phi = \circ$ and either $\mathit{lf}(q_1)$ or $\mathit{lf}(q_2)$ is not a bound variable, or $\phi = -$ and $\mathit{lf}(q_1)$ is not a bound variable; $\mathit{bci}_{\mathit{stp}}(q_1, \phi, q_2, R)$ exactly when $\mathit{bci}_{\mathit{stp}}(q_1, \phi, q_2)$. If $\mathit{bci}_{\mathit{stp}}(q_1, \phi, q_2)$ then $\mathit{matches}_{\mathit{stp}}(q_1, \phi, q_2, R_1)$ if and only if $\mathit{matches}_{\mathit{stp}}(q_1, \phi, q_2, R_2)$ for any R_1 and R_2. Then observe that if $\delta'^*(i, p\ell)$ is defined then $\delta'^*(i, p)$ is binder correspondence independent. Thus if $\delta'^*(i, p)$ is binder correspondence independent then determining if $p \in F$ can be done without tracking the binder correspondence at all.

Now consider p such that $\delta'^*(i, p)$ is binder correspondence dependent. By definition $\delta'^*(i, p) = \mathit{promote}_{\mathit{stp}}(q_{\mathsf{L}}, \phi, q_{\mathsf{R}}, R)$ for some q_{L}, ϕ, q_{R}, and R. The first three can be determined without tracking the binder correspondence. Consider what information is needed to determine if $p \in F$. Case 1, $\phi = \circ$: In this case $\mathit{lf}(q_{\mathsf{L}}) = \mathsf{bv}(q'_{\mathsf{L}}, \ell_{\mathsf{L}})$ and $\mathit{lf}(q_{\mathsf{R}}) = \mathsf{bv}(q'_{\mathsf{R}}, \ell_{\mathsf{R}})$ and $p \in F$ if and only $(q'_{\mathsf{L}}, q'_{\mathsf{R}}) \in R$. Case 2, $\phi = +$: In this case $\mathit{lf}(q_{\mathsf{R}}) = \mathsf{bv}(q'_{\mathsf{R}}, \ell)$. If there is no q'_{L} such that $(q'_{\mathsf{L}}, q'_{\mathsf{R}}) \in R$ then $p \in F$. If there is then $p \in F$ if and only if $q_{\mathsf{L}} \hookrightarrow^*_{\mathit{stp}} q'_{\mathsf{L}}$. Case 3, $\phi = -$: similar to the previous case.

My strategy for determining the above conditions is to compute facts of the form $q_1 \leftrightharpoons q_2$, $q \uparrow$, and $\uparrow q$ at the triples that are binder correspondence dependent. The meaning of $q_1 \leftrightharpoons q_2$ is that there is an R possible at the triple with $(q_1, q_2) \in R$; similarly $q \uparrow$ means there is an R possible at the triple with no q' such that $(q, q') \in R$; and $\uparrow q$ means there is an R possible at the triple with

no q' such that $(q', q) \in R$. Computing such facts is a simple dataflow problem. If (q_1, ϕ, q_2) is such that $lf(q_1) = lf(q_2) = \forall$ then $q_1 \leftrightharpoons q_2$ is generated, $q_1' \leftrightharpoons q_2'$ is propagated if $q_i \neq q_i'$, $q \uparrow$ is propagated if $q \neq q_1$, $\uparrow q$ is propagated if $q \neq q_2$, $q_1 \leftrightharpoons q$ where $q \neq q_2$ is changed to $\uparrow q$, and $q \leftrightharpoons q_2$ where $q \neq q_1$ is changed to $q \uparrow$. Function states propagate all facts.

In summary, the polynomial time algorithm computes the triples that are possible and checks that the binder correspondence independent ones match and sets aside the binder correspondence dependent ones. Then it sets up and solves the dataflow problem outlined above. Finally it uses the computed facts at the binder correspondence dependent triples to determine if they match or not. The algorithm is at worst $O(n^4)$ as quadratic dataflow facts need to be propagated to quadratic nodes, and the other phases are at least as good. It might be possible to do better by exploiting the scoping requirements of binders, but I have not explored this possibility.

5 Discussion

To put all the pieces together, all we need to do is define a way to go from types to binding-tree automata. In my previous work I defined exactly such a transformation—in particular given a type τ and a distinguishing environment η there is a binding-tree automata $automataof_\eta(\tau)$ constructable in linear time such that $treeof(\tau)_\eta = treeof(automataof_\eta(\tau))$. Combining that algorithm with the one in Section 4, gives a polynomial-time algorithm for deciding subtyping on the types themselves. The soundness and completeness of the rules and correctness of the algorithm means both that this algorithm is deciding subtyping according to the type rules, and that the type rules correspond to the tree interpretation of subtyping, which hopefully corresponds to our intuitive notion of what subtyping should be for the system under consideration.

Theorem 8. *If η is distinguishing then:*

$$B \vdash \tau_1 \leq \tau_2$$
$$\Leftrightarrow$$
$$L(subtype(automataof_\eta(\tau_1), automataof_\eta(B), automataof_\eta(\tau_2))) = \emptyset$$

This chapter considered the Kernel rule for subtyping forall quantified types. As previously mentioned, that rule is not the most general rule that is sound for such types. I believe that the definition of subtyping for binding trees and the typing rules can be modified for the most general rule and the soundness and completeness theorems can still be proven, but I have not done this. Nothing in the proofs is critically dependent on the bound being invariant. The construction of a DFA for subtyping, however, is critically dependent on the bound being invariant. In particular, to augment the construction for the full rule first requires tracking which side is the tightest bound (easy to do), but also requires figuring out the

binder correspondence after promoting to a bound, which requires saving the correspondence at the binding point leading to a linked list like correspondence information—no longer a finite set. Thus I believe this system is undecidable for similar reasons to full $F_{\leq}$.

There is another rule for subtyping forall quantified types that allows the bound to be contravariant but considers the variable unbounded in the body type. It is usually ignored as it leads to a lack of principal types. I believe that this rule could also be worked into my definition of trees, type rules, and automata construction—the variables bound by such quantifiers have no bounds, and so they act very similar to function types. I use a self quantifier in my object encoding that requires the body type to be covariant. As the body of the self quantifier is also the bound of the quantified variable, the typing rules cannot have an invariant bound. This variant with unbounded variables for checking the body is the most appropriate. The full rule is likely undecidable, and lack of principal types is avoided because the introduction form for self quantifiers includes a full type annotation, unlike for type lambdas that include only a partial type annotation.

Thus I believe that the results of this paper can be extended to handle other base types, first-order constructs of various flavours, other bound rules, and other second-order constructs like existentials and self quantifiers. Extending to higher-kinded systems is definitely future work and might not be possible. The first problem is that F_{ω} with equirecursive types has the simply-typed lambda calculus with general recursion at the type level, hence guaranteeing termination is probably a necessary first requirement.

References

[AC93] Amadio, R., Cardelli, L.: Subtyping recursive types. ACM Transactions on Progamming Languages and Systems 15(4), 575–631 (1993)

[CCH+89] Canning, P., Cook, W., Hill, W., Mitchell, J., Olthoff, W.: F-bounded quantification for object-oriented programming. In: 4th ACM Conference on Functional Programming and Computer Architecture, London, UK, pp. 273–280. ACM Press (September 1989)

[CG99] Colazzo, D., Ghelli, G.: Subtyping recursive types in kernel fun. In: 1999 Symposium on Logic in Computer Science, Trento, Italy, pp. 137–146 (July 1999)

[Gle00] Glew, N.: An efficient class and object encoding. In: ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Minneapolis, MN, USA. ACM Press (October 2000)

[Gle02a] Glew, N.: A theory of second-order trees. In: European Symposium on Programming 2002, Grenoble, France (April 2002)

[Gle02b] Glew, N.: A theory of second-order trees. Technical Report TR2001-1859, Department of Computer Science, Cornell University, 4130 Upson Hall, Ithaca, NY 14853-7501, USA (January 2002)

[Gle12] Glew, N.: Subtyping for F-bounded quantifiers and equirecursive types (extended version). arXiv:1202.2486 (February 2012), `http://arXiv.org/`

[GP04] Gauthier, N., Pottier, F.: Numbering matters: First-order canonical forms for second-order recursive types. In: 9th ACM SIGPLAN International Conference on Functional Programming, Snowbird, UT, USA, pp. 150–161. ACM Press (September 2004)

[KPS95] Kozen, D., Palsberg, J., Schwartzbach, M.: Efficient recursive subtyping. Mathematical Structures in Computer Science 5(1), 113–125 (1995)

[Pie94] Pierce, B.: Bounded quantification is undecidable. Information and Computation 112, 131–165 (1994)

Inferring Evolutionary Scenarios in the Duplication, Loss and Horizontal Gene Transfer Model

Paweł Górecki and Jerzy Tiuryn

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw,
Banacha 2, 02-097 Warszawa, Poland

Abstract. An H-tree is a formal model of evolutionary scenario. It can be used to represent any processes with gene duplication and loss, horizontal gene transfer (HGT) and speciation events. The model of H-trees, introduced in [26], is an extension of the duplication-loss model (DL-model). Similarly to its ancestor, it has a number of interesting mathematical and biological properties. It is, however, more computationally complex than the DL-model. In this paper, we primarily address the problem of inferring H-trees that are compatible with a given gene tree and a given phylogeny of species with HGTs. These results create a mathematical and computational foundation for a more general and practical problem of inferring HGTs from given gene and species trees with HGTs. We also demonstrate how our model can be used to support HGT hypotheses based on empirical data sets.

Keywords: Phylogenetic tree, Duplication-loss model, Rewrite system, Horizontal gene transfer.

1 Introduction

Horizontal gene transfer (HGT) is an evolutionary process in which genetic material is transferred between two unrelated or even distant organisms. For instance, the increased drug resistance in bacteria was proven to be spread by HGT for the first time in [42]. This biological phenomenon is considered to be an important factor in the evolution of a single cell organism like bacteria or archaea [23]. The presence of HGT, however, creates a serious difficulty in the process of inference of phylogenetic relationships from gene sequences, where the vertical transfer, that is, the transfer of genetic sequences from a parent to its offspring, is the main assumption.

In this paper, we are interested in the duplication-loss model, called DL-model [24, 20, 39, 40, 32, 44, 21, 33, 4, 47, 21, 3, 12, 5, 6, 1, 7, 18, 11, 9, 10, 31, 30, 16, 28] and its HGT extension [17, 38, 25, 35, 34, 19, 49, 46]. The DL model was originally developed to explain the differences between two incongruent gene and species trees. Under the assumption, that the species are 'containers' for the genes, one can generalize this property and embed a gene tree into a species tree by appropriate allocation of duplication and loss events in these trees. Such *embedding*, represented in the DL model by *reconciled tree* [43], reflects the common evolutionary history of the genes and their species, and addresses well the question of biological correctness. The reconciled tree,

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 83–105, 2012.

created in a process of gene-species reconciliation, has a number of interesting mathematical properties. One of the most important properties can be expressed in terms of gene duplication and loss events: the reconciled tree requires the minimal number of these events [14, 29]. In this sense, the model of reconciled trees is based on parsimony principle. Reconciled trees have been intensively studied in various theoretical and biological contexts in the last 20 years. The recent interest is also focused on non-parsimonious scenarios [26, 29, 27].

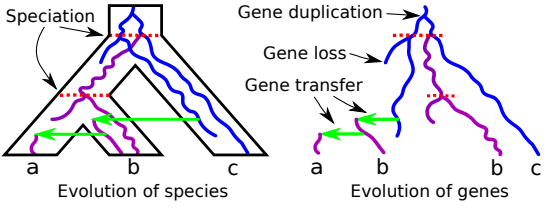


Fig. 1. Evolution of species and genes with HGTs

The DL-model can be extended by incorporating HGTs (see Fig. 1). In our previous paper [26], we introduced a formal framework of evolutionary scenario, called *H-tree*, which represents a common history of genes and species in the presence of macro-evolutionary events such as gene duplication and loss, speciation and HGT. H-trees can be used to represent *all possible evolutionary scenarios* under the above assumptions. Moreover, the model provides a detailed information on the evolution of a single gene family. For example, all the evolutionary events can be easily located in the species. Furthermore, an H-tree can be useful in the analysis of the evolution of genomes, for instance, in detecting whole genome duplications, multiple HGTs, etc.

As already mentioned, HGTs have been intensively studied in various theoretical contexts such as reticulation networks, tree rearrangements and reconciliation based models [45, 8, 36, 13, 41]. In all these contexts the general problem of HGT inference is computationally demanding [34, 15]. Our model of a species tree with HGTs is naturally compatible with the models introduced in [34].

In the present paper, we analyse the properties of H-trees that represent the common evolution of a given gene tree and a given *species graph*, that is, a species tree with HGTs. H-trees have a relatively simple rewrite system for finding an H-tree with a minimal cost (the number of duplication, loss and horizontal transfer events). As shown in [26] every H-tree has a normal form under the above mentioned system of rewrite rules. The main contribution of the present paper is to propose the notion of a scenario with respect to a gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$, and show a correspondence between scenarios and H-trees in normal form which are compatible with $\mathcal{G}$ and $\mathcal{S}$. Scenarios are just functions between leaves of certain trees, and therefore they represent relatively easy objects to work with. We then show how to compute the cost of an H-tree directly from the corresponding scenario. In this way we reduce the problem of finding H-trees with a minimal cost to finding the scenarios which minimize the cost.

This paper is organized as follows. In Section 2 we present an introduction to the model of H-trees. This section also contains preliminary definitions concerning trees, species graphs and others. Section 3 is devoted to setting up the main technical lemmas which will be used in the proofs of the main results, as well as the construction of an H-tree from a given gene tree $\mathcal{G}$, a species graph $\mathcal{S}$, and a scenario for $\mathcal{G}$ and $\mathcal{S}$. Section 4 contains the main results of the present paper: correspondence between scenarios and H-trees in normal form compatible with a gene tree and a species graph. It also contains formulas for computing the cost with respect to a given scenario. Section 5 presents the results of applying the machinery developed in the present paper to tackle the hypothesis of horizontal gene transfer from *Wolbachia* strains to mosquitoes, as described in [48]. Finally Section 6 contains several concluding remarks.

2 The Model of H-Trees

We start with several standard definitions related to graphs and sequences. Let G denote a directed graph $\langle V, E\rangle$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. A path in G is a sequence of nodes $p_1 p_2 \dots p_k$, denoted by $\boldsymbol{p}$, such that $\langle p_i, p_{i+1}\rangle \in E$, for $i = 1, 2, \dots, k-1$. The length of $\boldsymbol{p}$ is denoted by $|\boldsymbol{p}|$. If there is a directed path from p to q it is denoted $p \rightsquigarrow q$, and by $p \rightsquigarrow^+ q$ if the length of the path is greater than zero. If $\boldsymbol{p}$ is a sequence of k elements from the domain of a mapping f, then $f[\boldsymbol{p}]$ denotes the sequence $f(p_1)f(p_2)\dots f(p_k)$.

By $G(v)$ we denote the minimal subgraph of G that contains all nodes and edges that can be reached from a node v by a path. Let G be a *rooted directed acyclic graph (rDAG)*. Then G has a distinguished node v, called *a root*, denoted by $root(G)$, such that $G = G(v)$. A leaf in G is a node with no children. Non-leaf nodes are called *internal*. By Lf_G we denote the set of all leaves in G. Similarly, by Int_G we denote the set of all internal nodes of G. The set of all paths in rDAG G starting in the root and terminating in a leaf is be denoted by Π_G. If G and G' are graphs, then the notation $f\colon G \to G'$ denotes a mapping from the nodes of G into the nodes of G'.

A directed rooted graph B is *a tree*, if every node of B can be reached from $root(B)$ by a unique path. We call a tree B *binary* (or *semi-binary*) if each internal node of T has exactly (or at most, respectively) two children. Given a binary tree B and $A \subseteq \mathrm{Lf}_B$, the *restriction* of B to a set A is obtained by removing all leaves which are not in A and then suppressing all internal nodes of degree two. This restriction will be denoted by $B|A$.

By lca_B we denote the *lca-mapping* in B; that is, $\mathrm{lca}_B(a, b)$ is the least common ancestor of nodes a and b in B.

Let $\mathcal{I}$ be a set of species names. A *gene tree* is a binary tree whose leaves are labeled by the elements from $\mathcal{I}$. The labeling need not be one-to-one. *A species tree* is a gene tree whose leaves are uniquely labeled. In some cases, we use the notion of *semi-binary species tree*, that is, a semi-binary tree whose leaves are uniquely labeled by the species names.

Let $\mathcal{G}$ be a gene tree. By $L(\mathcal{G})$ we denote the set of all species names (labels) in $\mathcal{G}$. For a node v of $\mathcal{G}$, *a cluster* of v, denoted by m_v^T, is the set $L(\mathcal{G}(v))$. By $C(\mathcal{G})$ we denote the set of all clusters of $\mathcal{G}$. For convenience, when writing the set $\{x_1, x_2, \dots, x_k\}$ we usually skip the curly brackets and write $x_1 x_2 \dots x_k$. We use the standard parenthesis

notation for gene/species trees defined by the following set of productions $B \to a|(B, B)$ where $a \in \mathcal{I}$ (we omit straightforward details).

Let $\mathcal{F}$ be a family of subsets of $\mathcal{I}$. We call $\mathcal{F}$ an *$\mathcal{I}$-family* if, for each A and A' in $\mathcal{F}$ with $A \cap A' \neq \emptyset$, we have either $A \subseteq A'$ or $A' \subseteq A$. It can be easily proved that, for any $\mathcal{I}$-family $\mathcal{F}$, there exists a species tree B such that $\mathcal{F} \subseteq C(B)$. The converse also holds.

2.1 HGTs and Species Graphs

Modeling both HGTs and a phylogeny of species is a non-trivial task due to the potentially complex time dependencies that may occur between transfers. For example, a transfer cannot move genetic sequence backwards in time neither cross with another transfer. Here, we formally analyse the relations between transfers and introduce the notion of a species graph and an H-tree, and summarize their properties. Please refer to [26] for more details and examples.

A *species graph* is an ordered triple $\mathcal{S} = \langle V, E, H\rangle$ such that $B = \langle V, E\rangle$ is a semi-binary species tree and $H \subseteq V \times V$ satisfies the following conditions:

(HR1) For every $\gamma \in H$, nodes of γ are not on a path in B
(HR2) For every $\gamma \in H$, both nodes of γ have out-degree 1 (in B).
(HR3) No two edges in H have a node in common.
(HR4) Every node of V with out-degree 1 is contained in an edge from H
(HR5) The relation

$$\preccurlyeq = \{\langle \gamma_1, \gamma_2\rangle \in H^2 \mid \text{there exists a path in } B \text{ from a node of } \gamma_1 \text{ to a node of } \gamma_2\}$$

is a partial order in H.

Edges of H are called *horizontal transfers* and the relation $\preccurlyeq$ is called a *dependency relation*.

Observe that it follows from (HR1) that for all $v \in V$, we have $\langle v, v\rangle \notin H$ and that E and H are disjoint. It also follows from (HR5) that $\preccurlyeq$ is always reflexive and transitive. We will refer to the items related to a given species graph $\mathcal{S}$ by using it as an index. So, $H_{\mathcal{S}}$ denotes the set of horizontal transfers of $\mathcal{S}$, etc. The lca-mapping in $\mathcal{S}$, denoted by $\mathrm{lca}_{\mathcal{S}}$, and clusters are inherited from the semi-binary tree $B_{\mathcal{S}}$; for example, the cluster of a transfer node equals the cluster of its child.

The above definition of a species graph, although simpler than the one presented in [26], can be easily shown to be equivalent.

Let $\mathcal{F}$ be an $\mathcal{I}$-family and let H be a set. A mapping $\mathcal{A}\colon H \to \mathcal{F} \times \mathcal{F}$ which assigns pairs of disjoint subsets to elements of H is called a *transfer assignment*. Instead of writing $\mathcal{A}(\gamma) = \langle A, B\rangle$, we use, throughout the rest of this work, a notation $A \xrightarrow{\gamma} B \in \mathcal{A}$.

2.2 H-Trees and the Rewrite System

In this section, we introduce the crucial notion of an H-tree. The following symbols will be used: $\square$ (duplication), $\circ$ (loss), $\blacksquare$ (speciation), $\bullet$ (gene) and $\to$ (HGT).

An *H-tree* is a tuple $T = \langle \mathcal{I}, H, B, \mathcal{A}, \preceq\rangle$, where $\mathcal{I}$ is the set of species names, H is a set whose elements will be called transfers, $B = (V, E)$ is a binary rooted tree such

that each node of V is labeled by a non-empty subset of $\mathcal{I}$, function $\mathcal{A}\colon H \to \mathcal{F} \times \mathcal{F}$ is a transfer assignment over H such that $\mathcal{F}$ is an $\mathcal{I}$-family that contains all labels from B, there exists a partial order $\preceq^*$ such that for $\gamma, \gamma' \in H$ if $\mathcal{A}(\gamma) = \langle A_1, A_2 \rangle$ and $\mathcal{A}(\gamma') = \langle A'_1, A'_2 \rangle$, then:

(HD1) if $A_i \subset A'_j$, for some $i, j \in \{1, 2\}$, then $\gamma' \preceq^* \gamma$,
(HD2) if $A_i = A'_j$, for some $i, j \in \{1, 2\}$, then $\gamma' \preceq^* \gamma$ or $h \preceq^* \gamma'$,
(HD3) if a node of γ' is on the path in B from the root to a node of γ, then $\gamma' \preceq^* \gamma$,

and $\preceq$ is the least partial order $\preceq^*$ satisfying conditions (HD1)-(HD3). For $v \in V$, let Λ_v denote the label of v. V is divided into five disjoint subsets: $V_\bullet$, $V_\circ$, $V_\square$, $V_\blacksquare$, and $V_\rightarrow$, whose members are called *gene, loss, duplication, speciation* and *transfer* nodes, respectively. The labeling system is subject to the following conditions:

(L1) $V_\bullet \cup V_\circ = \mathrm{Lf}_B$,
(L2) the nodes of $V_\bullet$ are labeled by singletons,
(L3) the labels of a duplication node and its children are equal,
(L4) if v is a speciation node with two children a and b, then $\Lambda_a \cup \Lambda_b = \Lambda_v$ and $\Lambda_a \cap \Lambda_b = \emptyset$,
(L5) if v is a transfer node with two children a and b, then $\Lambda_a \cap \Lambda_b = \emptyset$ and for a certain $\gamma \in H$ we have $\Lambda_a \overset{\gamma}{\rightarrow} \Lambda_b \in \mathcal{A}$ (or with a and b interchanged). In this case we set $\Lambda_v = \Lambda_a$ and we call the edge $\langle v, b \rangle$ horizontal and we say that this edge is associated with transfer γ.

The least partial order on H which satisfies conditions (HD1)-(HD2) will be called *horizontal dependence on* $\mathcal{A}$. It has been shown in [26] (Lemma 2.4) that for any horizontal dependence $\preceq$ on $\mathcal{A}\colon H \to \mathcal{F} \times \mathcal{F}$: **(i)** there exists a species graph $\mathcal{S}$ such that $\mathcal{F} \subseteq C(\mathcal{S})$, **(ii)** there exists a bijection $\phi\colon H \to H_{\mathcal{S}}$ that preserves labels, and **(iii)** $\phi[\preceq] = \preccurlyeq_{\mathcal{S}}$.

The careful reader may have noticed that not every transfer $\gamma \in H$ has to be associated with a transfer node in T. For a similar reason not every species $a \in \mathcal{I}$ has to be present in B. Note that the label of each internal node of an H-tree can be easily reconstructed from the labeling of its leaves.

An H-tree is a formal representation of an evolutionary scenario. Each edge in the H-tree represents the evolution of a single gene lineage between two neighboring macroevolutionary events such as speciation, gene duplication, gene loss or HGTs. In addition, each horizontal edge represents a gene sequence that is horizontally transferred between two distinct species. Please note, that a transfer (that is, an element of H) can be composed of several horizontal edges. Therefore, any transfer can transmit more than one gene sequence at the same time. Furthermore, the dependencies between transfers are modeled by the relation $\preceq$. Thus, for any two transfers α and β, if $\alpha \preceq \beta$, then α is older than β. Please observe, that some transfers may be incomparable. In such a case, we do not assume any time dependencies between them. See Fig. 4 for examples of H-trees. With an H-tree T we associate *a cost* which is the total number of gene duplications, losses and horizontal edges in T. For example, the cost of $T^*_{1,2,2}$ presented in Figure 4 equals 6.

The condition (HD1)-(HD3) are required to filter improper scenarios. For example, if $\mathcal{I} = \{a, b\}$, $H = \{\alpha, \beta\}$ and the transfer assignment is defined as follows: $\mathcal{A}(\alpha) = \langle a, b \rangle$ and $\mathcal{A}(\beta) = \langle b, ac \rangle$, then there is no H-tree in which $\alpha \preceq \beta$.

Let T be an H-tree. Similarly to species graphs, we use the lower index to refer to the elements of the tuple defining H-tree, for example, H_T is the set of transfers in T. For convenience, sometimes we use T (a tuple) to denote the binary tree G_T (an element of the tuple) if this is not ambiguous. With this notation, we can naturally adopt already existing tree-like operations to H-trees. For example, for a node $v \in T$, by $T(v)$ we denote the H-subtree of T rooted at v; that is, $T(v) := \langle \mathcal{I}_T, H_T, B_T(v), \mathcal{A}_T, \preceq_T \rangle$.

In order to define operations on H-trees we introduce the notion of a pattern. Let $\mathcal{I}$ a set of names of species and H a set of transfers be fixed. Consider terms generated by the following grammar:

$$P \to \emptyset_H \mid a \mid A_\circ \mid (P,P)_{\blacksquare} \mid (P,P)_{\square} \mid (P,P)^{\gamma}_{\rightarrow}$$

where a ranges over $\mathcal{I}$, A ranges over subsets of $\mathcal{I}$, and γ ranges over H. The other symbols in the above grammar, except P, are terminals.

A word generated by the above grammar determines a set of H-trees in the following way:

- **(P1)** $\emptyset_H$ is the set of all empty H-trees (that is, H-trees $T = \langle \mathcal{I}, H, B, \mathcal{A}, \preceq \rangle$ with B being the empty tree),
- **(P2)** a determines the set of all H-trees $T = \langle \mathcal{I}, H, B, \mathcal{A}, \preceq \rangle$ with B being a one element tree whose gene node is labeled by $\{a\}$,
- **(P3)** $A_\circ$ is the set of all H-trees $T = \langle \mathcal{I}, H, B, \mathcal{A}, \preceq \rangle$ with B being a one element tree whose loss node is labeled by A,
- **(P4)** $(\mathcal{P}, \mathcal{Q})_{\blacksquare}$ is the set of H-trees $\langle \mathcal{I}, H, B, \mathcal{A}, \preceq \rangle$ such that there exist H-trees $\langle \mathcal{I}, H, B_p, \mathcal{A}, \preceq \rangle$ in $\mathcal{P}$ and $\langle \mathcal{I}, H, B_q, \mathcal{A}, \preceq \rangle$ in $\mathcal{Q}$, with roots p and q, respectively, such that the root of B is a speciation node with two children p and q and the trees B_p and B_q are rooted in H at p and q, respectively,
- **(P5)** the definition for $(\mathcal{P}, \mathcal{Q})_{\square}$ is similar to (P4), except that the root of B is a duplication node,
- **(P6)** the definition for $(\mathcal{P}, \mathcal{Q})^{\gamma}_{\rightarrow}$ is also similar to (P4), except that the root of B is a horizontal node and the edge connecting the root and q is horizontal and associated with γ.

Finally, we call a term $\mathcal{P}$ generated by the above grammar a *pattern* if it determines a non-empty set of H-trees.

Let us recall that in the above definition the sets $\mathcal{I}$ and H are fixed. Even though it may seem counterintuitive there is in general more than one empty H-tree as defined in (P1) – empty H-trees differ in choice of $\mathcal{A}$ and/or $\preceq$. For technical reasons we decided to choose this option. Same remark applies to objects defined in (P2) and (P3).

Note that if $\mathcal{P}$ is a pattern and an H-tree T is determined by $\mathcal{P}$, then for every species graph $\mathcal{S}$ which contains all transfers present in $\mathcal{P}$, the following tuple $\langle \mathcal{I}, H_{\mathcal{S}}, B_T, \mathcal{A}_{\mathcal{S}}, \preccurlyeq_{\mathcal{S}} \rangle$ is an H-tree. We will denote it by $\mathcal{P}[\mathcal{S}]$. The above definition is correct since it easily follows from (P1-P6) that if T_1 and T_2 are H-trees determined by the same pattern, then $B_{T_1} = B_{T_2}$.

For example, $T^*_{1,2,2}$ (see Fig. 4g) has the pattern: $\mathcal{P} = (((a, b_\circ)_{\blacksquare}, (c_\circ, b)^{\beta}_{\rightarrow})_{\blacksquare}, ((c, b)^{\beta}_{\rightarrow}, ab_\circ)_{\blacksquare})_{\square}$. Moreover, if $\mathcal{S}$ is the species graph presented in Fig. 4a, then $\mathcal{P}[\mathcal{S}]$ uniquely determines $T^*_{1,2,2}$.

Next we define the notion of compatibility between an H-tree on one hand side, and a species graph or a gene tree, on the other. Let T be an H-tree. It is said to be *compatible* with a species graph $\mathcal{S}$ if

(i) every label in B_T occurs as a cluster in $\mathcal{S}$,
(ii) there exists a bijection $\phi\colon H_T \to H_{\mathcal{S}}$ that preserves clusters and $\phi[\preceq_T] = \preceq_{\mathcal{S}}$.

T is said to be *compatible* with a gene tree $\mathcal{G}$ if $\mathcal{G}$ can be obtained from B_T by restricting its leaves to gene nodes only (that is, if $\mathcal{G} = B_T|V_\bullet$). We will sometimes denote the gene tree obtained in such a way from T by $gene(T)$. For $\mathcal{G}$ compatible with T we define a mapping $\Gamma_T\colon \mathrm{Lf}_{\mathcal{G}} \to V_\bullet$ such that, for a leaf g in $\mathcal{G}$, $\Gamma_T(g)$ is the corresponding gene node in T. Note that Γ_T is a bijection.

By $\mathfrak{C}(T)$ we denote the class of species graphs compatible with an H-tree T. Similarly, by $\mathfrak{C}(\mathcal{G}, \mathcal{S})$ we denote the class of H-trees compatible with a gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$. By $\Lambda(T)$ we denote the label of the root of T. An H-tree T compatible with $\mathcal{S}$ will be called $\mathcal{S}$-proper, if $\Lambda(F) = L(\mathcal{S})$.

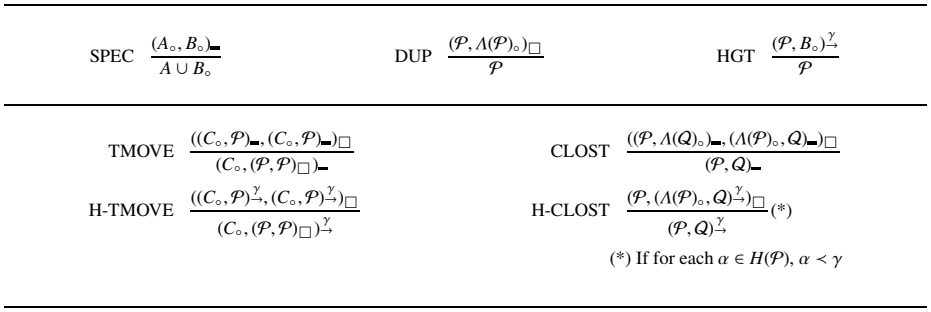


Fig. 2. Rewrite system. Rules of type I (top) and II (bottom). $\mathcal{R}, \mathcal{P}$ and $\mathcal{Q}$ are patterns.

In Figure 2 we propose a system of rules for transforming H-trees so that the cost of the tree is reduced after transformation. A rule, say R, is defined by $\frac{\mathcal{R}}{\mathcal{R}'}$, where $\mathcal{R}$ (premise) and $\mathcal{R}'$ (conclusion) are patterns. The mechanism of applicability of the rules is as follows: an H-tree T can be transformed into an H-tree T' by an application of a rule R to a node v in T if and only $T(v) \in \mathcal{R}$ and T' is constructed from T by replacing the subtree $T(v)$ in T by T'' such that $T'' \in \mathcal{R}'$. The defined transformation affects only the binary tree part of the H-trees. All other items of the H-trees remain unchanged. We denote by $R(T, v)$ the result of reduction. The node v is called *a redex*.

The subtree S of $T(v)$ is called *principal* for a rule R applied to a node v if:

(i) if R is one of the rules DUP, HGT, TMOVE, or H-TMOVE then $S \in \mathcal{P}$,
(ii) if R is one of the rules CLOST, or H-CLOST, then $S \in \mathcal{P}$, or $S \in \mathcal{Q}$.

Moreover, for H-CLOST, which is not symmetric, we distinguish between *left* (pattern $\mathcal{P}$) and *right* (pattern $\mathcal{Q}$) principal tree. The application of H-CLOST is conditional.

A reduction step is of *type I* (or *type II*) if the applied rule is of type I (or type II, respectively).

We conclude this section by stating the fundamental properties of H-trees which were proved in [26]. Let $\sim$ be the least equivalence relation on the class of all H-trees which contains the relation of reductions. Thus, if $T \sim T'$, then T can be transformed into T' by applying rules zero or more times in any direction. An H-tree T is said to be: **(i)** *lost* if it is equivalent to an H-tree with a pattern $\Lambda(T)_\circ$, **(ii)** *in normal form* if T has no redexes, **(iii)** *semi-normal* if T has no redexes of type I, and finally **(iv)** *fat* if T is a semi-normal tree such that for every H-tree T', if T' can be reduced into T, then T' is not semi-normal. Hence fat trees are semi-normal, maximal in the class of semi-normal trees with respect to the reduction relation.

Theorem 1. ([26]) *Let T and T' be H-trees. Then: **(i)** (confluency) there exists a unique H-tree T^* (in normal form) such that every sequence of reductions, which starts in T and terminates in normal form, yields T^*, **(ii)** (soundness) if T and T' are semi-normal and equivalent then $gene(T) = gene(T')$ and $\mathfrak{C}(T) \subseteq \mathfrak{C}(T')$ **(iii)** T and T' are equivalent iff $T^* = T'^*$ and **(iv)** T^* is the unique tree with the minimal cost in the set of all trees that are equivalent to T.*

Similar results have been proven for the system of reversed type II rules restricted to semi-normal H-trees.

Theorem 2. ([26]) *Let T and T' be semi-normal H-trees. Then: **(i)** (confluency) there exists a unique fat tree T^f such that every sequence of reversed reductions of type II, which starts in T and terminates in a fat tree, yields T^f, **(ii)** T and T' are equivalent iff $T^f = T'^f$ and **(iii)** T^f is the unique tree with the maximal cost in the set of all semi-normal trees that are equivalent to T.*

3 H-Trees and Scenarios

In this section, we investigate some properties of H-trees that are reconstructed from a given gene tree and a given species graph. In other words, we present a detailed analysis of the set $\mathfrak{C}(\mathcal{G}, \mathcal{S})$, where $\mathcal{G}$ is a gene tree and $\mathcal{S}$ is a species graph.

3.1 Reconstruction of Fat Trees

We start with a simple property that can be used to classify fat trees: if T is fat, then each child of a duplication node in T is either a duplication node or is the root of the tree with a pattern $(B_{1\circ}, (B_{2\circ}, \ldots (B_{k\circ}, a) *_k \ldots) *_2) *_1$. where $a \in \mathcal{I}$, $k \geq 0$ and, for $i = \{1, 2, \ldots, k\}$, $*_i \in \{\rule{1em}{0.4em}, \stackrel{\gamma_i}{\rightarrow}\}$. An H-tree with the above pattern is called *a chain tree* and the path connecting the root with the only gene node in a chain tree is called *a core path*. In other words, this property states that each fat tree can be divided into two parts: the top part consisting of duplication nodes and the rest with chain trees. For a gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$, we denote by $\mathfrak{Fat}(\mathcal{G}, \mathcal{S})$ the set of all fat trees from $\mathfrak{C}(\mathcal{G}, \mathcal{S})$.

Let $\boldsymbol{p}$ be a path in $\mathcal{S}$ that terminates in a leaf. By $\mathfrak{cp}(\boldsymbol{p})$ we denote a sequence of nodes which is obtained from $\boldsymbol{p}$ by removing **(I)** each transfer's start node if its termination node does not belong to $\boldsymbol{p}$ and **(II)** each transfer's termination node. We call $\mathfrak{cp}(\boldsymbol{p})$ *a c-path*.

Lemma 1. *Let $\mathcal{S}$ be a species graph. Then, there is a one-to-one correspondence between c-paths in $\mathcal{S}$ and chain trees compatible with $\mathcal{S}$. Moreover, for any c-path $\boldsymbol{c}$ in $\mathcal{S}$, there exists exactly one path $\boldsymbol{p}$ in $\mathcal{S}$ such that $p_1 = c_1$, $p_{|\boldsymbol{p}|} = c_{|\boldsymbol{c}|}$ and $\mathfrak{cp}(\boldsymbol{p}) = \boldsymbol{c}$.*

Proof. We define two mappings between c-paths and chain trees from which the correspondence can be deduced. (*Mapping $\tau_{\mathcal{S}}$: c-paths $\rightarrow$ chain trees*). Let $\boldsymbol{q}$ be a c-path of the length k. Then there exists a path $\boldsymbol{p}$ in $\mathcal{S}$ such that $\mathfrak{cp}(\boldsymbol{p}) = \boldsymbol{q}$. Assume that $\boldsymbol{p}$ terminates in a leaf labeled by a. We claim that the procedure presented below defines a chain tree compatible with $\mathcal{S}$: (i) $C_k = a$ if $i = k$, (ii) $C_i = (m_{q_i\circ}, C_{i+1})^{h}_{\rightarrow}$ if $i < k$ and q_i is the start node of a transfer h (please note that if $h = \langle q_i, z\rangle$ then then $m_z = m_{q_{i+1}} = \Lambda(C_{i+1})$ and both nodes of h are in $\boldsymbol{p}$), (iii) $C_i = ((m_{q_i} \setminus m_{q_{i+1}})_\circ, C_{i+1})_{\blacksquare}$ if $i < k$ and q_i is a non-transfer node. Let $\tau_{\mathcal{S}}(\boldsymbol{q}) := C_1[\mathcal{S}]$. Observe that m_{q_i} is the label of i-th node in the core path of $\tau_{\mathcal{S}}(\boldsymbol{q})$. The correctness of the above procedure follows from the definition of c-path.

(Mapping $\sigma_{\mathcal{S}}$: chain trees $\rightarrow$ c-paths). Let C be a chain tree compatible with $\mathcal{S}$ and $\boldsymbol{c}$ the core path of C. We define a path $\boldsymbol{q}$ in $\mathcal{S}$. **(A0)** Set q_k to be the leaf labeled by $a \in \Lambda_{c_k}$. Assume that $i < k$: **(A1)** if c_i is a speciation node, then q_i is a non-transfer node whose cluster is Λ_{c_i}, **(A2)** if c_i is a γ-transfer node then q_i is the start node of a transfer $\phi(\gamma)$ where ϕ is the bijection from the definition of compatibility. We claim that $\boldsymbol{q}$ is a c-path. It remains to show that there exists a path from which $\boldsymbol{q}$ is obtained. For cases A0-A2 we define a sequence $\boldsymbol{p^i}$. **(A0)** Let $\boldsymbol{p^k}$ be one element sequence q_k. **(A1)** $\boldsymbol{p^i}$ is the unique path from q_i to q_{i+1} in T concatenated with the sequence $\boldsymbol{p^{i+1}}$. **A2)** In this case $\Lambda_{c_{i+1}} = m_v = m_{q_{i+1}}$ where $\phi(\gamma) = \langle q_i, v\rangle$. Please observe that v is ancestor of q_{i+1}. Let $\boldsymbol{p^i}$ be the shortest path connecting v and q_{i+1} in $\mathcal{S}$ concatenated with $\boldsymbol{p^{i+1}}$.

Finally we set $\sigma_{\mathcal{S}}(C)$ to be $\boldsymbol{p^1}$ with removed duplicates. It should be clear that, for each c-path $\boldsymbol{q}$ in $\mathcal{S}$, $\sigma_{\mathcal{S}}(\tau_{\mathcal{S}}(\boldsymbol{q})) = \boldsymbol{q}$, and, for each chain tree C compatible with $\mathcal{S}$, $\tau_{\mathcal{S}}(\sigma_{\mathcal{S}}(C)) = C$.

The notion of c-path will be useful in proving combinatorial properties of fat H-trees in the context of a given species graph $\mathcal{S}$. Therefore, later on we use $\sigma_{\mathcal{S}}$, defined in the proof of Lemma 1, to denote the transformation of chain trees into the c-paths. Please note that the result presented in the proof is much stronger. Indeed, for each chain tree C, $|C| = |\sigma_{\mathcal{S}}(C)|$ and the label of i-th node of the core path of C equals the cluster of i-th node of the c-path $\sigma_{\mathcal{S}}(C)$ for any $i = 1, 2, \ldots, |C|$.

The second part of the Lemma follows immediately from the above proof. □

Recall that $\Pi_{\mathcal{S}}$ denotes the set of all paths in $\mathcal{S}$ from the root to leaves.

Lemma 2. *For a given species graph $\mathcal{S}$ with n leaves and h transfers: $|\Pi_{\mathcal{S}}| \leq n2^h$. Moreover, there exists a species graph $\mathcal{S}$ with h horizontal gene transfers such that $|\Pi_{\mathcal{S}}| \geq \phi^h$ where $\phi = \frac{1+\sqrt{5}}{2}$.*

Proof. First let us assume that $n > h$. It can be proved, that the maximal number of paths has the left species graph $\mathcal{S}$ shown in Fig. 3, where the subtree marked by star has $n - h$ leaves. For each node s of $\mathcal{S}$, we show the number of possible paths that can reach s starting from the root. Summing over leaves: $|\Pi_{\mathcal{S}}| = \sum_{i=0}^{h-1} 2^i + (n-h)2^h \leq n2^h$.

Now we analyse the other case. First we assume that $n = 2$. It is easy to see that the species graph $\mathcal{T}$ with alternating transfers, presented in Fig. 3, has the maximal size of $|\Pi_{\mathcal{S}}|$ over all species graphs with two leaves. Let a_h denote the sequence of possible

paths of $\mathcal{T}$ with h transfers. Then, $a_0 = 2$, $a_1 = 3$ and $a_h = a_{h-1} + a_{h-2}$ for $h \geq 2$. Thus, $a_n = F_{n+3}$, where F_n is the n-th Fibonacci number. Finally, $a_h = [\frac{\phi^{h+3}}{\sqrt{5}}]$, where $\phi = \frac{1+\sqrt{5}}{2}$ and $[x]$ is the nearest integer function.

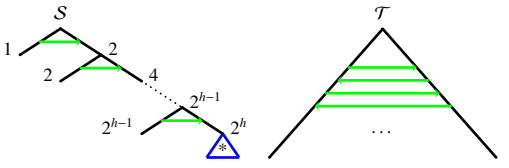


Fig. 3. Species graphs $\mathcal{S}$ and $\mathcal{T}$. Horizontal gene transfers of $\mathcal{T}$ are alternating.

For the general case, where $n \leq h$, it can be proved that the maximal number of paths has the species graph $\mathcal{S}$ from Fig. 3 in which the subtree marked by a star has the topology of tree $\mathcal{T}$. Thus, this subtree has $h - n + 2$ alternating transfers. Finally: $|\Pi_{\mathcal{S}}| = \sum_{i=0}^{n-3} 2^i + 2^{n-2}[\frac{\phi^{h-n+5}}{\sqrt{5}}] \leq 2^{n-2} - 1 + 2^{h-n+3}2^{n-2} \leq n2^h$.

The second part of the lemma follows immediately from the above proof. □

We proved that the number of chain trees in the species graph $\mathcal{S}$ can be exponential in the number of HGTs in $\mathcal{S}$. Having this, we can state a similar conclusion for $\mathfrak{Fat}(\mathcal{G}, \mathcal{S})$, where $\mathcal{G}$ is a gene tree, by presenting a simple algorithm: **(i)** let $\{l_1, \ldots, l_n\}$ be a set of leaves in $\mathcal{G}$, **(ii)** for each cluster X in $\mathcal{S}$ and each sequence of chain trees $\langle C_1, \ldots, C_n \rangle$ such that C_i and $\mathcal{S}$ are compatible, $X = \Lambda(C_i)$ and the cluster of l_i equals the label of the only gene node in C_i, construct a fat H-tree: take $\mathcal{G}$ and set elements of $\mathrm{Int}_{\mathcal{G}}$ to be duplications labeled by X, then replace each l_i by C_i.

By $\mathfrak{Fat}^*(\mathcal{G}, \mathcal{S})$, we denote the set of all $\mathcal{S}$-proper H-trees in $\mathfrak{Fat}(\mathcal{G}, \mathcal{S})$. For instance, in Figure 4g, $\mathfrak{Fat}^*(\mathcal{G}, \mathcal{S}) = \{F_{i,j,k} \mid i \in \{1, 2, 3\}, j, k \in \{1, 2\}\}$.

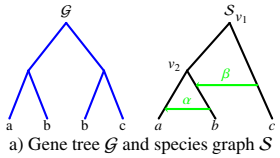
The following lemma states the connection between $\mathcal{S}$-proper trees and other trees in $\mathfrak{Fat}(\mathcal{G}, \mathcal{S})$.

Lemma 3. *Let $F \in \mathfrak{Fat}(\mathcal{G}, \mathcal{S})$.*

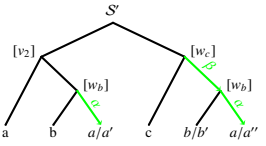
(i) There exists $\tilde{F} \in \mathfrak{Fat}^*(\mathcal{G}, \mathcal{S})$ such that $\tilde{F}$ can be transformed by a sequence of TMOVE reductions into F' and F is a subtree of F'.

(ii) There exists $\tilde{F} \in \mathfrak{Fat}^*(\mathcal{G}, \mathcal{S})$ such that the normal form of F is a subtree of the normal form of $\tilde{F}$.

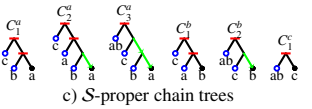
Proof. We first prove (i). Let us start with the construction of $\tilde{F}$. Observe that the root of each chain tree in F is labeled by $\Lambda(F)$. Let $\boldsymbol{s}$ be the maximal sequence of non-transfer nodes connecting the root of $\mathcal{S}$ and the non-transfer node in $\mathcal{S}$ whose cluster is $\Lambda(F)$. Please note that both the path and the non-transfer node are unique. It should be clear that, for each C, the sequence $\boldsymbol{q} = s_1 \ldots s_{k-1}\sigma_{\mathcal{S}}(C)$ is a c-path in $\mathcal{S}$. Moreover, $C^* = \sigma_{\mathcal{S}}^{-1}(\boldsymbol{q})$ is the chain tree that contains C as a subtree and $\Lambda(C^*) = L(\mathcal{S})$. Now we can define $\tilde{F}$: **(i)** take F and replace each chain tree C in F by C^* and **(ii)** set the label of each duplication node to be $\Lambda(\mathcal{S})$.



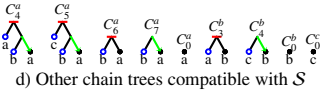
a) Gene tree $\mathcal{G}$ and species graph $\mathcal{S}$



b) Extended species tree $\mathcal{S}'$



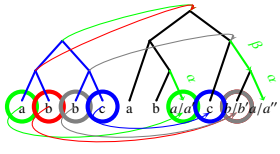
c) $\mathcal{S}$-proper chain trees



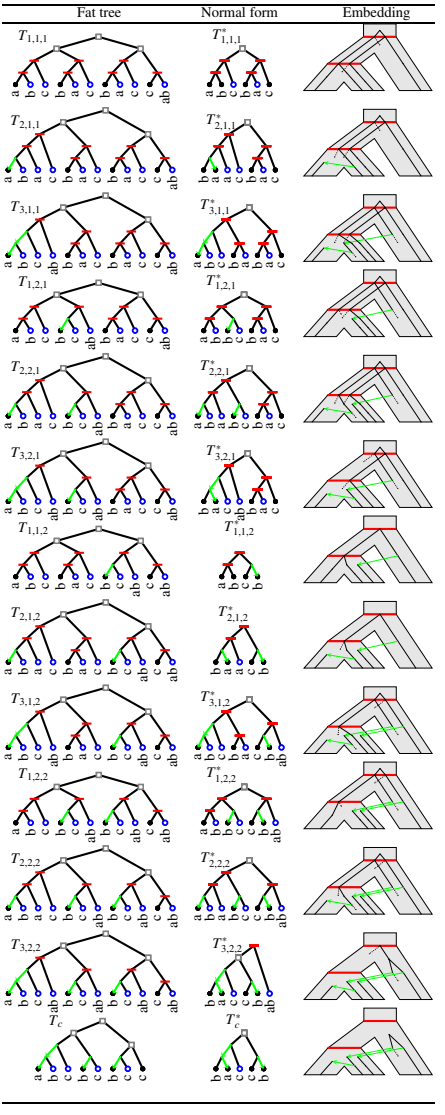
d) Other chain trees compatible with $\mathcal{S}$

Chain tree	c-path	Clusters in $\mathcal{S}$	Leaf in $\mathcal{S}'$
$\mathcal{S}$-proper chain trees			
C_1^a	v_1, v_2, a	abc, ab, a	a
C_2^a	v_1, v_2, w_b, a	abc, ab, b, a	a'
C_3^a	v_1, w_c, w_b, a	abc, c, b, a	a''
C_1^b	v_1, v_2, b	abc, ab, b	b
C_2^b	v_1, w_c, b	abc, c, b	b'
C_1^c	v_1, c	abc, c	c
Non $\mathcal{S}$-proper chain trees			
C_4^a	v_2, w_b, a	ab, b, a	
C_5^a	w_c, w_b, a	c, b, a	
C_6^a	v_2, a	ab, a	
C_7^a	w_b, a	b, a	
C_0^a	a	a	
C_3^b	v_2, b	ab, b	
C_4^b	w_c, b	c, b	
C_0^b	b	b	
C_0^c	c	c	

e) Properties of chain trees compatible with $\mathcal{S}$



f) Scenario and mapping for $T_{2,2,2}$ and $T^*_{2,2,2}$



g) $\mathfrak{Fat}(\mathcal{S}, \mathcal{G})$, corresponding normal forms and embeddings.

Fig. 4. An example of a gene tree and a species graph with corresponding H-trees. (a) Gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$ (see also Fig. 1). (b) The extended species tree with $\Upsilon_{\mathcal{S}}$ mapping (in square brackets). See section 3.2 for more details. (c)-(e) Chain trees related to $\mathcal{S}$. (f) An example of a scenario. See section 3 for more details. (g) All fat and normal form H-trees compatible with $\mathcal{S}$ and $\mathcal{G}$. $F_{i,j,k}$ is a fat tree containing chain trees: C_i^a, C_j^b, C_k^b and C_1^c. F_c contains chain trees: C_5^a, C_4^b and C_0^c. Note that the evolutionary scenario presented in Fig. 1 is represented here by $T^*_{3,2,1}$.

Now we prove, for $F \in \mathfrak{Fat}(\mathcal{G}, \mathcal{S})$, $\tilde{F}$ can be transformed by a sequence of TMOVE reductions into

$$F' = (B^1_\circ, (B^2_\circ, \ldots (B^{k-1}_\circ, F)\_ \ldots)\_)\_, \tag{1}$$

where $B_i = A^i \setminus A^{i+1}$ and A_i is the cluster of s_i. This property can be proved by induction on the size of $\mathcal{G}$. The main step is depicted in Figure 5. We skip the rest of the proof for brevity.

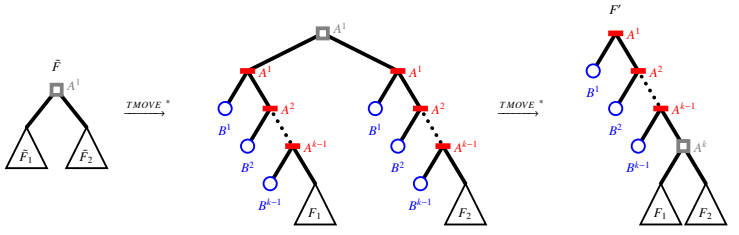


Fig. 5. The main step of induction; here $F = (F_1, F_2)\square$

For the proof of (ii) take $\tilde{F}$ from (i) of the present lemma. $\tilde{F}$ reduces to F' (defined by (1)). It is clear that the first $k-1$ non-lost nodes in F' (that is, labeled by $A^1, A^2, \ldots, A^{k-1}$, respectively) cannot be a redex of any rule. Hence, if F^* is a normal form of F, then $(B^1_\circ, (B^2_\circ, \ldots (B^{k-1}_\circ, F^*)\_ \ldots)\_)\_$ is a normal form of F' and (by uniqueness) $\tilde{F}$. □

It follows from Theorem 1 and 2 that the number of trees in normal form compatible with a given gene tree $\mathcal{G}$ and a given species graph $\mathcal{S}$ equals $|\mathfrak{Fat}(\mathcal{G}, \mathcal{S})|$. Obviously, it is also equal to the number of $\sim$-equivalence classes in $\mathfrak{C}(\mathcal{G}, \mathcal{S})$. See Fig. 4g for more examples. Note that T^*_c (the normal form of T_c) is a subtree of $T^*_{3,2,2}$ (the normal form of $T_{3,2,2}$).

In general $\tilde{F}$ from Lemma 3 is not uniquely determined. Note that the prefix $s_1 s_2 \ldots s_n$ from the proof of (i) of Lemma 3 is chosen arbitrarily. In particular, the prefix may contain transfer nodes. However, in our example (Fig. 4) there is exactly one such prefix in $T^*_{3,2,2} = (ab_\circ, T^*_c)$. We skip details.

Let us assume that T_1 and T_2 are H-trees such that $gene(T_1) = gene(T_2)$ and $\mathfrak{C}(T_1) = \mathfrak{C}(T_2)$. From this section, we know that, in general, these trees are not equivalent.

3.2 Extended Species Trees

A species graph that contains many HTGs has usually a complex structure where the paths can intersect in many ways. This property creates a serious difficulty in the reconstruction of H-trees that are compatible with the species graph and a given gene tree. To overcome it, we 'unfold' the species graph into the tree-like representation. Based on such representation we will be able to solve the reconstruction problem in a more natural way. The unfolded species graph is called here *an extended species tree*.

In several contexts, for instance species graphs or H-trees, some edges of a directed graph are related to HGTs. Therefore, we introduce a useful generic notation for such edges. Let H is the set of horizontal edges in a directed graph G. We use the notation

$a \xrightarrow{\gamma} b$ if $\langle a, b\rangle \in H$ is associated with a transfer γ. A non-transfer edge $\langle a, b\rangle$ is denoted by $a \xrightarrow{-} b$. A generic notation $\twoheadrightarrow$ will be used for any directed edge. Reversed arrows denote the reversed relations.

We extend the standard nested parenthesis notation for trees by defining a new production for horizontal edges: $T \to (T, T)^{\gamma}_{\rightarrow}$. This production denotes the construction of a tree similarly to (T, T) with the exception that the edge connecting the root with the right subtree is horizontal and associated with a transfer γ.

For the species graph $\mathcal{S}$, the extended species tree, denoted by $\mathcal{S}'$, is defined by $\epsilon_{\mathcal{S}}(root(\mathcal{S}))$, where

$$\epsilon_{\mathcal{S}}(s) = \begin{cases} a & \text{if } s \text{ is the leaf in } \mathcal{S} \text{ labeled by } a, \text{ (i)} \\ (\epsilon_{\mathcal{S}}(p), \epsilon_{\mathcal{S}}(q)) & \text{if } p \xleftarrow{-} s \xrightarrow{-} q,\ q \neq p, \quad \text{(ii)} \\ (\epsilon_{\mathcal{S}}(p), \epsilon_{\mathcal{S}}(q))^{\gamma}_{\rightarrow} & \text{if } p \xleftarrow{-} s \xrightarrow{\gamma} q, \quad \text{(iii)} \\ \epsilon_{\mathcal{S}}(v) & \text{if } z \xrightarrow{\gamma} s \xrightarrow{-} v \quad \text{(iv)} \end{cases} \tag{2}$$

Now we can define a mapping $\Upsilon_{\mathcal{S}} : \mathcal{S}' \to \mathcal{S}$. For a node s' in $\mathcal{S}'$, $\Upsilon_{\mathcal{S}}(s') = s$ such that $root(\epsilon_{\mathcal{S}}(s)) = s'$ and s is not a transfer termination node. Please note that s is unique. An example of a species graph and its extended species trees with $\Upsilon_{\mathcal{S}}$ mapping is depicted in Figure 4b, where the nodes in square brackets denotes the mapping.

We start with Lemmas that connects the properties of paths in $\mathcal{S}$ and $\mathcal{S}'$. Let $\mathfrak{rp}(T, g)$ denote the path starting in the root of a rooted tree T and terminating in the leaf g.

Lemma 4. *Let $\mathcal{S}$ be a species graph. There is a one-to-one correspondence between $\Pi_{\mathcal{S}}$ and $Lf_{\mathcal{S}'}$ determined by the definition of an extended species tree.*

Proof. It follows from the definition of $\mathcal{S}'$ that a path from $\Pi_{\mathcal{S}}$ uniquely determines a leaf in $\mathcal{S}'$. Let s' be a leaf in $\mathcal{S}'$. Now, we show that there exists $\boldsymbol{s} \in \Pi_{\mathcal{S}}$ such that $\mathfrak{cp}(\boldsymbol{s}) \subseteq \Upsilon_{\mathcal{S}}[\mathfrak{rp}(\mathcal{S}', s')]$. Observe that if $\mathfrak{rp}(\mathcal{S}', s')$ contains exactly one node from a horizontal edge then it must be the start node of this edge. We claim that removing mappings of such nodes from $\Upsilon_{\mathcal{S}}[\mathfrak{rp}(\mathcal{S}', s')]$ yields a c-path in $\mathcal{S}$ that starts in the root. By Lemma 1 we can reconstruct $\boldsymbol{s}$ from the c-path. In addition, we have $\Upsilon_{\mathcal{S}}[\mathfrak{rp}(\mathcal{S}', s')] \subseteq \boldsymbol{s}$. We omit details. □

From the proof of Lemma 4, we define a bijection $\mathfrak{I}_{\mathcal{S}} \colon \Pi_{\mathcal{S}} \to \mathrm{Lf}_{\mathcal{S}'}$ such that, for a path $\boldsymbol{s}$ in $\Pi_{\mathcal{S}}$, $\mathfrak{I}_{\mathcal{S}}(\boldsymbol{s})$ is the corresponding leaf in $\mathcal{S}'$.

3.3 Scenarios and the Trees in Normal Form

Let $\mathcal{G}$ be a gene tree and $\mathcal{S}$ a species graph. A *scenario* is a mapping from the leaves of $\mathcal{G}$ into leaves of extended species tree $\mathcal{S}'$ which preserves labels. A scenario $\xi \colon \mathrm{Lf}_{\mathcal{G}} \to \mathrm{Lf}_{\mathcal{S}'}$ naturally extends to the least common ancestor mapping $\hat{\xi} \colon \mathcal{G} \to \mathcal{S}'$ such that $\hat{\xi}(g) = \xi(g)$ if g is a leaf and $\hat{\xi}(g) = \mathrm{lca}_{\mathcal{S}'}(\hat{\xi}(a), \hat{\xi}(b))$ if $g \in \mathrm{Int}_{\mathcal{G}}$ with two children a and b. We show that scenarios are crucial for the reconstruction of trees in normal form from a given gene tree $\mathcal{G}$ and a given species graph $\mathcal{S}$.

First, we define a mapping $\psi_{\mathcal{G},\mathcal{S}',\xi}$. This definition is by induction on the structure of $\mathcal{G}$ and $\mathcal{S}'$. Formally, $\psi_{\mathcal{G},\mathcal{S}',\xi}$ takes two arguments: a subtree G of $\mathcal{G}$ and a subtree S of $\mathcal{S}'$ such that $\hat{\xi}(root(G)) \in S$ and returns a pattern. If $G = S = a$, then $\psi_{\mathcal{G},\mathcal{S}',\xi}(G, S) = a$

(a is a label). Let $s = root(S)$, $g = root(G) \in \mathrm{Int}_{\mathcal{G}}$ with two children p and q. Then (the subscripts are omitted in ψ):

$$\psi(G,S) = \begin{cases} (\psi(G(p),S),\psi(G(q),S))_{\square} & \text{if } \hat{\xi}(g) = s = \hat{\xi}(p), \quad \text{(DP)} \\ (\psi(G(p),S(a),\psi(G(q),S(b)))_{\blacksquare} & \text{if } \hat{\xi}(p) \leftsquigarrow a \overset{-}{\leftarrow} s = \hat{\xi}(g) \\ & \text{and } s \overset{-}{\rightarrow} b \rightsquigarrow \hat{\xi}(q), \quad \text{(SP)} \\ (\psi(G(p),S(a),\psi(G(q),S(b)))^{\gamma}_{\rightarrow} & \text{if } \hat{\xi}(p) \leftsquigarrow a \overset{-}{\leftarrow} s = \hat{\xi}(g) \\ & \text{and } s \overset{\gamma}{\rightarrow} b \rightsquigarrow \hat{\xi}(q), \quad \text{(HT)} \\ (m^{S}_{b\circ},\psi(G,S(a)))_{\blacksquare} & \text{if } \hat{\xi}(g) \leftsquigarrow a \overset{-}{\leftarrow} s \overset{-}{\rightarrow} b \neq a, \text{(SP-L)} \\ (m^{S}_{b\circ},\psi(G,S(a)))^{\gamma}_{\rightarrow} & \text{if } \hat{\xi}(g) \leftsquigarrow a \overset{\gamma}{\leftarrow} s \overset{-}{\rightarrow} b \neq a, \text{(HT-L)} \\ \psi(G,S(a)) & \text{if } \hat{\xi}(g) \leftsquigarrow a \overset{-}{\leftarrow} s \overset{\gamma}{\rightarrow} b \neq a. \text{(N-HT)} \end{cases} \tag{3}$$

Finally, let $\Psi_{\xi}(\mathcal{G},\mathcal{S}) = \psi_{\mathcal{G},\mathcal{S}',\xi}(\mathcal{G},\mathcal{S}')[\mathcal{S}]$.

4 Main Results

Now we can state the main results of the present paper which establish a relationship between scenarios and H-trees in normal form compatible with a given gene tree and a species graph.

Theorem 3. *Let $\mathcal{G}$ be a gene tree and $\mathcal{S}$ a species graph such that $\emptyset \neq L(\mathcal{G}) \subseteq L(\mathcal{S})$. Then*

(i) For every scenario ξ for $\mathcal{G}$ and $\mathcal{S}$, $\Psi_{\xi}(\mathcal{G},\mathcal{S})$ defines an $\mathcal{S}$-proper H-tree in normal form, compatible with $\mathcal{G}$.
(ii) Conversely: for every $\mathcal{S}$-proper tree T in normal form compatible with $\mathcal{G}$, there exists a scenario ξ for $\mathcal{G}$ and $\mathcal{S}$ such that $\Psi_{\xi}(\mathcal{G},\mathcal{S}) = T$.
(iii) Moreover, for every tree T in normal form compatible with $\mathcal{G}$ and $\mathcal{S}$ (not necessarily $\mathcal{S}$-proper), there exists a scenario ξ for $\mathcal{G}$ and $\mathcal{S}$ such that T is a subtree of $\Psi_{\xi}(\mathcal{G},\mathcal{S})$.

Proof. First we prove (i). Observe that the size of the trees G and S in recursive applications of (3) decreases. Hence the procedure terminates. We claim that, for each subtree G of $\mathcal{G}$ and each subtree S of $\mathcal{S}'$ satisfying $\hat{\xi}(root(G)) \in S$: **(i)** $\psi_{\mathcal{G},\mathcal{S},\xi}(G,S)[\mathcal{S}]$ is an H-tree compatible with G and $\mathcal{S}$, and **(ii)** $\Lambda(\psi_{\mathcal{G},\mathcal{S},\xi}(G,S)[\mathcal{S}]) = A$, where A is the cluster of $\Upsilon_{\mathcal{S}}(root(S))$. We omit the technical proof of this properties. Finally, $\Upsilon_{\mathcal{S}}(root(\mathcal{S}')) = root(\mathcal{S})$, therefore, by (ii), $\Psi_{\xi}(\mathcal{G},\mathcal{S})$ is an $\mathcal{S}$-proper H-tree.

To prove that $\Psi_{\xi}(\mathcal{G},\mathcal{S})$ is in normal form we show that there is no redex in $\psi_{\mathcal{G},\mathcal{S},\xi}(G,S)[\mathcal{S}]$ for G and S subtrees of $\mathcal{G}$ and $\mathcal{S}'$, respectively, satisfying $\hat{\xi}(root(G)) \in S$. We omit the subscript in ψ.

Observe that $\psi(G,S)$ cannot be a lost leaf. Hence, there is no pattern $(A_{\circ},B_{\circ})_{*}$ in $\psi(\mathcal{G},\mathcal{S}')$. Therefore, there is no redex of SPEC in $\psi(G,S)$. Other rules of type I can be solved similarly.

For the rules of type II we start with a general observation. Suppose that $\Psi_{\xi}(\mathcal{G},\mathcal{S})$ contains a redex of a rule of type II. Thus, there exist G and S such that the root of $\psi(G,S)$ is the redex and $\psi(G,S) = (R_1,R_2)_{\square}$. Such a pattern matches only (DP)

case in (3). Hence, $\hat{\xi}(root(G)) = root(S) = \hat{\xi}(p)$, where p is a child of the root of G. For TMOVE reduction we have $R_1 = (\mathcal{P}, C_\circ)_{\blacksquare}$ and $R_2 = (Q, C_\circ)_{\blacksquare}$. From (DP) we have $\psi(G(p), S) = (R_i, C_\circ)_{\blacksquare}$, for some $i \in \{1, 2\}$. Such a pattern matches only (SP-L) in (3) which requires $\hat{\xi}(root(G(p))) = \hat{\xi}(p) \neq root(S)$, a contradiction. CLOST and H-TMOVE reductions are similar to TMOVE. The reduction by H-CLOST is more complex due to its asymmetry. We have two cases where $\psi(G(p), S)$ is either **(i)** the pattern of the left principal tree of H-CLOST or **(ii)** $(C_\circ, Q)^{\gamma}_{\rightarrow}$, where Q is the pattern of the right principal tree. The second case leads to a contradiction similarly to the previous rules. Let us consider case (i). Obviously, $\psi(G(q), S)$ contains an edge associated with γ. We show that the same holds for $\psi(G(q), S)$. Let $s = root(S)$. Then, there is exactly one horizontal edge associated with γ in S: $s \overset{\gamma}{\rightarrow} s'$. Moreover, there exists at least one leaf a in $G(p)$ such that $s' \rightsquigarrow \hat{\xi}(a)$. Let v be the first node on the path from g to a such that $\hat{\xi}(v) \neq s$. Then, by (DP) and (HT-L): $\psi(G(p), S) = (\ldots \psi(G(v), S(s'))^{\gamma}_{\rightarrow} \ldots)_{\square}$. We proved that both subtrees of $\psi(G, S)$ contain a transfer γ. This is a contradiction with the condition from H-CLOST as defined in Figure 2. This completes the proof of (i). For the proof of (ii) and (iii) we will pause for now and prove one more lemma. □

Let T be an H-tree compatible with a species graph $\mathcal{S}$. We define a mapping $\Xi_T : T \rightarrow \mathcal{S}$. For a node v in T, the cluster of $\Xi_T(v)$ equals the label of v and: **(i)** if v is a leaf or a speciation node, then $\Xi_T(v)$ is the (unique) non-transfer node in $\mathcal{S}$, **(ii)** if v is a duplication node with two children p and q, then $\Xi_T(v)$ is the least common ancestor of $\Xi_T(p)$ and $\Xi_T(q)$ in the semi-binary tree of $\mathcal{S}$, **(iii)** if v is a γ-transfer node, then $\Xi_T(v)$ is the start node of the transfer $\phi(\gamma)$, where ϕ is the bijection from the definition of compatibility. Easy proof of the correctness of this definition is omitted.

Figure 6 depicts two equivalent H-trees $T^*_{3,1,2}$ and $F_{3,1,2}$ (see Fig. 4g). For each node of both H-trees the corresponding node from $\mathcal{S}$ is presented.

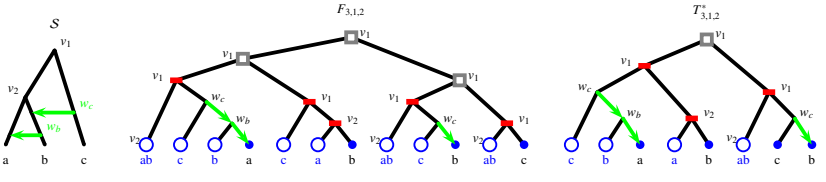


Fig. 6. A species graph $\mathcal{S}$ and a mapping Ξ_T for nodes of $T \in \{F_{3,1,2}, T^*_{3,1,2}\}$

Lemma 5. *Let $\mathcal{S}$ be a species graph and T be an $\mathcal{S}$-proper H-tree. Then,*

(i) For each gene node t in T, there exists exactly one path $\Delta_{\mathcal{S}}(t)$ in $\Pi_{\mathcal{S}}$ such that

$$\mathfrak{cp}(\Delta_{\mathcal{S}}(t)) \subseteq rd(\Xi_T[\mathfrak{rp}(T, t)]) \subseteq \Delta_{\mathcal{S}}(t), \tag{4}$$

where $rd(\boldsymbol{r})$ is the sequence obtained from $\boldsymbol{r}$ by removing all duplicates; that is, each $r_i r_{i+1} \ldots r_{i+k}$ is replaced with r_i if $r_i = r_{i+1} = \cdots = r_{i+k}$ and $k > 0$.

(ii) For an $\mathcal{S}$-proper H-tree T let $\pi(\mathcal{S}, T)$ be the multiset $\{\Delta_{\mathcal{S}}(t) \mid t \in V^T_{\bullet}\}$. Then T and T' are equivalent if and only if $\pi(\mathcal{S}, T) = \pi(\mathcal{S}, T')$.

(iii) Let $\mathcal{G}$ be a gene tree such that $\emptyset \neq L(\mathcal{G}) \subseteq L(\mathcal{S})$. If $T = \Psi_\xi(\mathcal{G}, \mathcal{S})$ for some scenario $\xi : Lf_{\mathcal{G}} \to Lf_{\mathcal{S}'}$, then for every gene node t of T we have

$$\mathfrak{cp}(\Lambda_{\mathcal{S}}(t)) \subseteq rd(\Upsilon_{\mathcal{S}}[\mathfrak{rp}(\mathcal{S}', \xi(\Gamma_T^{-1}(t)))]) \subseteq \Lambda_{\mathcal{S}}(t).$$

Proof. We start with a proof of (i). First we prove the statement for semi-normal trees. Observe that a reduction does not change the set of gene nodes in an H-tree. We show, in addition, that each reduction preserves not only a gene node but also the corresponding sequence $\Lambda_{\mathcal{S}}(t)$ with the property (4).

We proceed by induction on the length of a sequence of reductions of type II. Suppose, that T is fat. Then, t corresponds to a chain tree C in T (see Section 3.1). Hence, $\mathfrak{rp}(T, t) = d_1 d_2 \dots d_k c_1 c_2 \dots c_m$, where, $\boldsymbol{d}$ is a sequence of duplication nodes labeled by $L(\mathcal{S})$ and $\boldsymbol{c}$ is the core path of C. Note that for each i, $\Xi_T(d_i) = \Xi_T(c_1) = root(\mathcal{S})$ and there is no duplication node in $\boldsymbol{c}$. It follows immediately from Lemma 1 that if C is a chain tree compatible with a species graph $\mathcal{S}$. Then $\sigma_{\mathcal{S}}(C) = \Xi_C[\boldsymbol{c}]$, where $\boldsymbol{c}$ is the core path of C that $\sigma_{\mathcal{S}}(C) = \Xi_T[\boldsymbol{c}] = \mathrm{rd}(\Xi_T[\boldsymbol{dc}]$. Set $\Lambda_{\mathcal{S}}(t)$ to be the unique path (by Lemma 1) that contains $\sigma_{\mathcal{S}}(C)$.

Assume that the statement holds if T is obtained from the fat tree by a sequence of k reductions of type II. Let $T' = R(T, v)$ by a rule R of type II. Then, by the induction hypothesis, for a gene node t in T, we have the unique sequence $\Lambda_{\mathcal{S}}(t)$ satisfying (4). Without loss of generality we assume that the redex is in $\mathfrak{rp}(T, t)$, otherwise $\mathfrak{rp}(T, t) = \mathfrak{rp}(T', t)$. Then, t is a leaf in P where P in one of the principal trees of this reduction. Observe that there is no change in the paths if R = H-CLOST and P is the left principal tree. In other cases the path that connects v with $p = root(P)$ consists of three nodes: v, w and p such that $a = \Xi_T(v) = \Xi_T(w)$ and $b = \Xi_T(p)$. Reductions CLOST and the remaining case of H-CLOST transform $\Xi_T(vwp) = aab$ into ab. Obviously, (4) is satisfied with the same sequence $\Lambda_{\mathcal{S}}t$. Reductions TMOVE and H-TMOVE transform aab into axb, where $x = \mathrm{lca}_{\mathcal{S}}(b, c)$, $c = \Xi_T(q)$ and q is the root of the second principal tree. Please note that the middle node after transformation is a duplication. Hence, either $x = b$ (a trivial case) or $x = c \neq b$. In the latter case, x is the start node of a transfer. Thus, $x \in \Lambda_{\mathcal{S}}(t)$ but $x \notin \mathfrak{cp}(\Lambda_{\mathcal{S}}(t))$. Hence, $\Xi_T[\mathfrak{rp}(T, t)] \subset \Xi_T[\mathfrak{rp}(T', t)] \subseteq \Lambda_{\mathcal{S}}(t)$. This completes the proof for semi-normal trees.

To complete the proof for any tree, we need to extend the above proof by induction on the length of a sequence of reductions of type I in the reversed direction. We start with a semi-normal tree and a similar induction hypothesis. The proof for the reversed HGT is similar to TMOVE where the transformation introduces a new transfer node. We skip the details. Other cases are trivial. This completes the proof of (i) of the present lemma.

The proof of part (ii) of the lemma follows immediately from the property that reductions preserve $\pi(\mathcal{S}, T)$ (see the above proof of (i)).

For the proof of (iii) we observe that each gene node t in T corresponds to a gene leaf $g = \Gamma_T^{-1}(t)$ and a leaf $s' = \xi(\Gamma_T^{-1}(t))$ in $\mathcal{S}'$. Consider a sequence of applications of ψ that generates the path $\mathfrak{rp}(T, t)$, that is, $\psi(\mathcal{G}(g_1), \mathcal{S}'(v_1))$, $\psi(\mathcal{G}(g_2), \mathcal{S}'(v_2))$, …, $\psi(\mathcal{G}(g_k), \mathcal{S}'(v_k))$, where $v_1 = root(\mathcal{S}')$, $g_1 = root(\mathcal{G})$, $v_k = s'$, and $g_k = g$. It should be clear that $\boldsymbol{v} \subseteq \mathfrak{rp}(\mathcal{S}', s')$. Such a path in $\mathcal{S}'$ uniquely determines a path $\mathfrak{I}_{\mathcal{S}}^{-1}(s')$ in $\mathcal{S}$. We show that $\mathfrak{I}_{\mathcal{S}}^{-1}(s') = \Lambda_{\mathcal{S}}(t)$. For each i, let t_i be the root of a subtree of $\Psi_\xi(\mathcal{G}, \mathcal{S})$

with a pattern $\psi(\mathcal{G}(g_i), \mathcal{S}'(v_i))$. Then, **(SQ)** $\Xi_T(t_i) = \Upsilon_{\mathcal{S}}(v_i)$, if v_i does not satisfy the following condition: $v_i \in \mathrm{Int}_{\mathcal{S}'}$ and $\hat{\xi}(g_i) \leftsquigarrow a \overset{-}{\leftarrow} v_i \overset{\gamma}{\rightarrow} b \neq a$ (see N-HT in (3)). Please note that the right side of (SQ) is an element of $\mathrm{rd}(\Upsilon_{\mathcal{S}}[\mathfrak{tp}(\mathcal{S}', \xi(\Gamma_T^{-1}(t)))])$, while the left side defines an element of $\mathrm{rd}(\Xi_T[\mathfrak{tp}(T, t)])$. By (i) and the uniqueness, we conclude that (iii) also holds. □

Now we can conclude the proof of Theorem 3.

Proof of Theorem 3 (ii) and (iii). For the proof of (ii) let $\xi = \mathfrak{I}_{\mathcal{S}} \circ \Delta_{\mathcal{S}} \circ \Gamma_T$ and $T' = \Psi_{\xi}(\mathcal{G}, \mathcal{S})$. By Lemma 5 (iii), $\pi(\mathcal{S}, T) = \pi(\mathcal{S}, T')$. Then, by Lemma 5 (ii), T and T' are equivalent. Finally, by uniqueness of the normal form, $T = T'$.

Part (iii) follows from part (ii) if $\Lambda(T) = L(\mathcal{S})$. Other cases follow from Lemma 3. This completes the proof of Theorem 3. □

We conclude this section by presenting formulas for computing the cost of trees in normal form. For the gene losses, we define a relation $\diagdown$ on the set of nodes of $\mathcal{S}'$. For s and r in $\mathcal{S}'$, let $s \diagdown r$ if and only if **(i)** $s \rightsquigarrow^+ r$, and **(ii)** if $s \overset{\gamma}{\rightarrow} s'$, then $s' \rightsquigarrow r$.

Let ξ be a scenario for a gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$. For each node g, we define a non-negative integer $\mathrm{loss}_{\xi,g}$ as follows: $\mathrm{loss}_{\xi,g} = 0$ if $g \in \mathrm{Lf}_{\mathcal{G}}$. Otherwise, if $g \in \mathrm{Int}_{\mathcal{G}}$ with two children p and q then

$$\mathrm{loss}_{\xi,g} = \begin{cases} d(\hat{\xi}(g), \hat{\xi}(p)) + 1 & \text{if } \hat{\xi}(q) = \hat{\xi}(g) \diagdown \hat{\xi}(p), \\ d(\hat{\xi}(g), \hat{\xi}(p)) + d(\hat{\xi}(g), \hat{\xi}(q)) & \text{otherwise.} \end{cases}$$

where $d(s, s') = |\{t \ : \ s \rightsquigarrow^+ t \diagdown s'\}|$.

Also, let $\mathrm{loss}_{\xi,0} = |\{t \mid t \diagdown M(root(\mathcal{G}))\}|$. Now we can give the formula for the number of gene duplication and gene loss events.

Proposition 1. *Let ξ be a scenario for a gene tree $\mathcal{G}$ and a species graph $\mathcal{S}$. Then,*

(i) The number of duplications in $\Psi_{\xi}(\mathcal{G}, \mathcal{S})$ is given by

$$|\{g \mid \hat{\xi}(g) = \hat{\xi}(p), \text{ where } p \text{ is a child of } g \text{ in } \mathcal{G}\}|.$$

(ii) The number of gene losses in $\Psi_{\xi}(\mathcal{G}, \mathcal{S})$ is given by

$$loss_{\xi,0} + \sum_{g \in \mathcal{G}} loss_{\xi,g}.$$

Proof. The proof of (i) follows easily from (DP) in (3).

For the proof of (ii), let $T = \mathcal{S}'(\hat{\xi}(root(\mathcal{G})))$. We show that $\mathrm{loss}_{\xi,0}$ equals the number of gene losses in the first applications of ψ until T is reached as the second argument of $\psi_{\mathcal{G},\mathcal{S},\xi}$; that is, $\Psi_{\xi}(\mathcal{G}, \mathcal{S}) = (A_{\circ}^1, \ldots (A_{\circ}^k, \psi(\mathcal{G}, T))_* \ldots)_*[\mathcal{S}]$, where $k = \mathrm{loss}_{\xi,0}$. Consider a path $\boldsymbol{s}$ in $\mathcal{S}'$ starting in $root(\mathcal{S}')$ and terminating in $\hat{\xi}(root(\mathcal{G}))$. For each $i < |\boldsymbol{s}|$, **(i)** if $s_i \diagdown \hat{\xi}(root(\mathcal{G}))$, then either s_i is a non-transfer node (case (SP-L) in (3)) or $s_i \rightarrow s_{i+1}$ (case (HT-L)), otherwise **(ii)** s_i is a transfer node and $s_i \overset{-}{\rightarrow} s_{i+1}$ (case (N-HT)). Hence, a gene loss is present if and only if $s_i \diagdown s$.

Each node g in $\mathcal{G}$, naturally and uniquely corresponds to a node g^* in $\Psi_{\xi}(\mathcal{G}, \mathcal{S})$ such that $T(g^*)$ has pattern $\psi(\mathcal{G}(g), \mathcal{S}'(\hat{\xi}(g)))$. For $g \in \mathrm{Int}_{\mathcal{G}}$ with two children p and q, consider

two paths $g^* p_1 p_2 \ldots p_k p^*$ and $g^* q_1 q_2 \ldots q_m q^*$ in $\Psi_\xi(\mathcal{G}, \mathcal{S})$. We show that $\text{loss}_{\xi,g} = k+m$. Without loss of generality, we may assume that $g = root(\mathcal{G})$ and $\hat{\xi}(g) = root(\mathcal{S}')$. There are three cases **(1)**-**(3)** dependent on the mappings of g, p and q. **(1)** $\hat{\xi}(p) \neq \hat{\xi}(g) = \hat{\xi}(q)$ - case (DP) in (3). $T(q)$ has pattern $\psi(\mathcal{G}(q), \mathcal{S})$, and hence $m = 0$. For the second child, we have a sequence of (SP-L), (HT-L) and (N-HT) cases, similarly to (i)-(ii). Hence, the number of gene losses equals $|\{t : \hat{\xi}(g) \rightsquigarrow t \diagdown \hat{\xi}(p)\}|$. From the definition of d, $\hat{\xi}(g) \notin \{t : \hat{\xi}(g) \rightsquigarrow^+ t \diagdown \hat{\xi}(p)\}$. One gene loss is required, when $\hat{\xi}(g) \diagdown \hat{\xi}(p)$. Hence, if $\hat{\xi}(g) \diagdown \hat{\xi}(p)$, then $\text{loss}_{\xi,g} = d(\hat{\xi}(g), \hat{\xi}(p)) + 1$. Otherwise, $\text{loss}_{\xi,g} = d(\hat{\xi}(g), \hat{\xi}(p)) + d(\hat{\xi}(g), \hat{\xi}(q))$. **(2)** $\hat{\xi}(p) = \hat{\xi}(g) = \hat{\xi}(q)$ - case (DP). Here $k = m = 0$. Easy proof is omitted. **(3)** $\hat{\xi}(p) \neq \hat{\xi}(g) \neq \hat{\xi}(q)$ - case (SP) or (HT). Then, similarly to **(2)** with child p, the number of gene losses for p equals $|\{t : \hat{\xi}(g) \rightsquigarrow t \diagdown \hat{\xi}(p)\}|$. And hence, $d(\hat{\xi}(g), \hat{\xi}(p)) = k$. Analogously, $m = d(\hat{\xi}(g), \hat{\xi}(q))$. Finally, $\text{loss}_{\xi,g} = d(\hat{\xi}(g), \hat{\xi}(p)) + d(\hat{\xi}(g), \hat{\xi}(p))$. □

The problem of computing the scenario with minimal cost for a given gene tree and a given species graph can be solved by a dynamic programming algorithm along the lines presented in [25]. In the experimental section (Section 5) we will apply this adjusted version of the previously published algorithm to the biological data.

5 HGT Analysis

As an illustration of our model, we present the reconciliation analysis of the hypothesis of HGT from [48]. The authors used several complete genome sequences of the mosquitoes *Ae. aegypti*, *An. gambiae*, *Culex quinquefasciatus*, and of *Wolbachia pipientis* strains infecting *D. melagonaster* (*w*Mel) and *Cx. quinquefasciatus* (*w*Pip) to show that the SGS genes (encoding salivary gland surface proteins) had been horizontally transferred between these taxa. The results presented in [48] supported rather unusual direction of an ancient transfer from mosquito to *Wolbachia* strains. Here, we demonstrate the analysis of both directions of the transfer.

HGT from Wolbachia to mosquitoes. This hypothesis is summarized in Fig. 7 and Fig. 8. Figure 7 depicts a species graph $\mathcal{S}$ with one transfer as proposed in [48] and the phylogeny $\mathcal{G}$ of SGS genes. Please note, that there are four normal form H-trees compatible with $\mathcal{S}$ and $\mathcal{G}$. Here, we present only two embeddings E_1 and E_2 for the H-trees in normal form with the lowest reconciliation cost. The embedding E_1 is supported by the analysis from [48]. However, its reconciliation cost is 15 (2 HGTs, 5 gene duplications and 8 gene losses) is higher than the cost of E_2 ($14 = 1 + 4 + 9$, which is minimal in the set of all trees in normal form that are compatible with $\mathcal{S}$ and $\mathcal{G}$). Thus, in terms of reconciliation, the scenario represented by E_1 is less likely than E_2.

The authors of [48] supported also other HGT scenario related to the presence of the weak edge in $\mathcal{G}$ as indicated in Fig. 7. This variant of $\mathcal{G}$, denoted by $\mathcal{G}'$, is depicted in Fig. 8 together with two best scoring (normal form) embeddings of $\mathcal{G}'$ into $\mathcal{S}$. Now, the proposed HGT scenario from [48] is E'_1 which is in agreement with the reconciliation cost analysis. Indeed, the embedding E'_1 with two HGTs is optimal with cost 10 while the remaining embeddings require more events. For instance, E_2 requires 11 events and other two scenarios (not presented here) have cost 19.

HGT from Wolbachia to mosquitoes. The opposite direction of the transfer of the SDS genes was suggested by several authors, for example, [2, 37]. Such a hypothesis

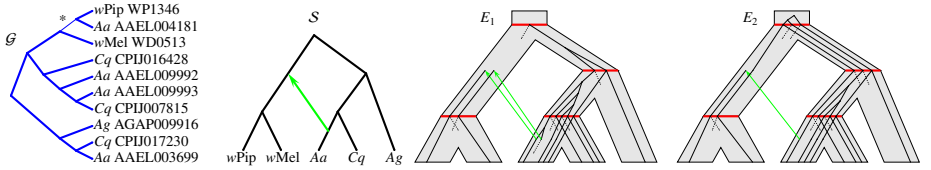


Fig. 7. An example of HGT between three mosquitoes (*Ae. aegypti*=*Ae*, *An. gambiae*=*Ag*, *Culex quinquefasciatus*=*Cq*) and two *Wolbachia* strains *w*Mel and *w*Pip. $\mathcal{G}$ is a gene tree (partially adopted from [48]) that represents phylogenetic relationships of mosquito SGS genes and their *Wolbachia* homologs. The edge marked with a star is short (weak) in $\mathcal{G}$. $\mathcal{S}$ is a phylogeny of mosquitoes [22] and *Wolbachia* strains with one hypothetical HGT with the direction as suggested in [48]. E_1 and E_2 represents two of four possible normal form embeddings of $\mathcal{G}$ into $\mathcal{S}$. E_2 is the optimal scenario in terms of the reconciliation cost (that is, the total number of evolutionary events). Please note that all the trees are rooted. Branch lengths have no meaning in this picture.

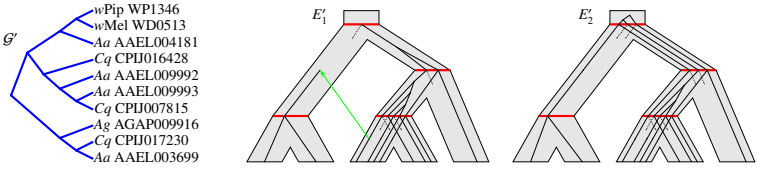


Fig. 8. $\mathcal{G}'$ - a gene tree obtained from $\mathcal{G}$ (see Fig. 7) by changing the topology around weak edge (indicated in Fig. 7 by a star). E'_1 (1 HGT, 4 gene duplications and 5 gene losses) and E'_2 (4 gene duplications and 7 gene losses) - two best scoring embeddings of $\mathcal{G}'$ into $\mathcal{S}$.

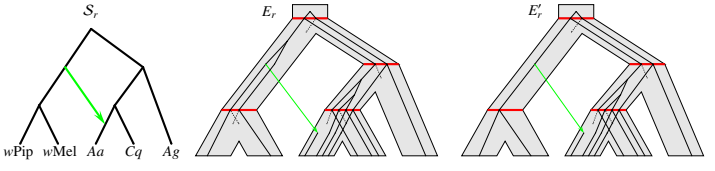


Fig. 9. $\mathcal{S}_r$ - a species graph obtained from $\mathcal{S}$ (Fig. 7) with the reversed transfer. E_r (1 HGT, 4 gene duplications and 6 gene losses) is the optimal embedding compatible with $\mathcal{S}_r$ and $\mathcal{G}$ (Fig. 7). E'_r (1 HGT, 3 gene duplications and 4 gene losses) is the optimal embedding compatible with $\mathcal{S}_r$ and $\mathcal{G}'$ (Fig. 8).

is depicted in Fig. 9 with the the species graph $\mathcal{S}_r$. For both gene trees $\mathcal{G}$ and $\mathcal{G}'$ from the previous figures the number of scenarios equals 16. The optimal embedding for $\mathcal{G}$ and $\mathcal{S}_r$ is E_r with the reconciliation cost 11. Other embeddings compatible with $\mathcal{G}$ have costs at least 15. Similarly, the optimal embedding for $\mathcal{G}'$ and $\mathcal{S}_r$ is E'_r with the cost 8 while the remaining scenarios compatible with $\mathcal{G}'$ have costs at least 11.

In summary, we conclude that the reconciliation analysis supports the prokaryote-to-eukaryote HGT as depicted in Figure 8.

6 Conclusions

This paper contains several theoretical and practical results on the problem of modelling an evolutionary scenario in the presence of macro-evolutionary events such as HGT, gene duplications, gene losses and speciations. The formal model of such scenarios, called H-trees, was introduced in our previous paper [26]. In the present paper, we solved a more practical problem of inference, that is, the problem of reconstruction of such evolutionary scenarios from a given gene tree and a given species graph. In addition, we answer a number of related questions. For example, we show how to compute costs or optimal scenarios.

Moreover, our model can be used as a basis for the definition of a reconciled tree with HGTs. This would be similar to the corresponding definition of reconciled trees in the duplication-loss model [43]. The point of departure for the new definition would be our definition of function Ψ (see (3)). For the sake of space we omit the details here. For example, a reconciled tree with HGTs that corresponds to the mapping given in Fig. 4f is depicted in Fig. 10.

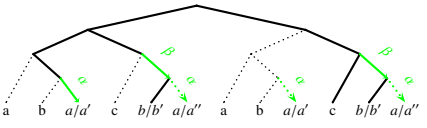


Fig. 10. Reconciled tree that corresponds to the mapping presented in Fig. 4f

We believe that these results are useful in practical terms and can be applied in testing empirical hypotheses of HGTs as demonstrated in the last section of this paper.

Acknowledgements. This research was partly supported by grant N N301 065236 of Polish Ministry of Science and Higher Education.

For Dexter Kozen

This part is written by JT. It is for me a great pleasure and honor to contribute to this volume on the occasion of 60-th birthday of Dexter Kozen. I have realized that I have known Dexter for 30 years, which is for us more than half of our lives as adults. For all these years Dexter and I were good friends. I have very good memories of our working together on various research problems in logic (I regret that it has stopped since I moved my research interest from TCS to Computational Biology), or writing a book (great time during several meetings with Dexter and David Harel at the Weizmann Institute in Israel), or simply doing jointly a real physical work (be it putting wallpaper in Dexter's apartment in New York City, or building a playground in Ithaca, or working on my old summer house in the woods of North-East Poland). Each of these activities has left very good memories and I sincerely hope there will be more in the years to come.

References

[1] Akerborg, O., Sennblad, B., Arvestad, L., Lagergren, J.: Simultaneous bayesian gene tree reconstruction and reconciliation analysis. PNAS 106(14), 5714–5719 (2009)

[2] Arca, B., Lombardo, F., Valenzuela, J.G., Francischetti, I.M.B., Marinotti, O., Coluzzi, M., Ribeiro, J.A.C.: An updated catalogue of salivary gland transcripts in the adult female mosquito, anopheles gambiae. J. Exp. Biol. 208, 3971–3986 (2005)

[3] Arvestad, L., Berglund, A.-C., Lagergren, J., Sennblad, B.: Bayesian gene/species tree reconciliation and orthology analysis using mcmc. Bioinformatics 19(Suppl. 1), i7–i15 (2003)

[4] Arvestad, L., Berglund, A.-C., Lagergren, J., Sennblad, B.: Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution. In: RECOMB, pp. 326–335 (2004)

[5] Arvestad, L., Lagergren, J., Sennblad, B.: The gene evolution model and computing its associated probabilities. J. ACM 56(2) (2009)

[6] Bansal, M.S., Eulenstein, O.: The multiple gene duplication problem revisited. Bioinformatics 24(13), i132–i138 (2008)

[7] Bansal, M.S., Eulenstein, O., Wehe, A.: The gene-duplication problem: near-linear time algorithms for nni-based local searches. IEEE/ACM Trans. Comput. Biol. Bioinform. 6(2), 221–231 (2009)

[8] Bansal, M.S., Gogarten, J.P., Shamir, R.: Detecting Highways of Horizontal Gene Transfer. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 109–120. Springer, Heidelberg (2010)

[9] Bansal, M.S., Burleigh, J.G., Eulenstein, O., Wehe, A.: Heuristics for the Gene-Duplication Problem: A $\Theta(n)$ Speed-Up for the Local Search. In: Speed, T., Huang, H. (eds.) RECOMB 2007. LNCS (LNBI), vol. 4453, pp. 238–252. Springer, Heidelberg (2007)

[10] Bansal, M.S., Eulenstein, O.: An $\Omega(n^2/\log n)$ Speed-Up of TBR Heuristics for the Gene-Duplication Problem. In: Giancarlo, R., Hannenhalli, S. (eds.) WABI 2007. LNCS (LNBI), vol. 4645, pp. 124–135. Springer, Heidelberg (2007)

[11] Bansal, M.S., Eulenstein, O.: The Gene-Duplication Problem: Near-Linear Time Algorithms for NNI Based Local Searches. In: Măndoiu, I., Wang, S.-L., Zelikovsky, A. (eds.) ISBRA 2008. LNCS (LNBI), vol. 4983, pp. 14–25. Springer, Heidelberg (2008)

[12] Berglund-Sonnhammer, A.-C., Steffansson, P., Betts, M.J., Liberles, D.A.: Optimal gene trees from sequences and species trees using a soft interpretation of parsimony. J. Mol. Evol. 63(2), 240–250 (2006)

[13] Boc, A., Makarenkov, V.: New Efficient Algorithm for Detection of Horizontal Gene Transfer Events. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 190–201. Springer, Heidelberg (2003)

[14] Bonizzoni, P., Della Vedova, G., Dondi, R.: Reconciling a gene tree to a species tree under the duplication cost model. Theor. Comput. Sci. 347(1-2), 36–53 (2005)

[15] Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Ann. Comb. 8, 409–423 (2004)

[16] Burleigh, J.G., Bansal, M.S., Wehe, A., Eulenstein, O.: Locating large-scale gene duplication events through reconciled trees: implications for identifying ancient polyploidy events in plants. J. Comput. Biol. 16(8), 1071–1083 (2009)

[17] Charleston, M.: Jungles: a new solution to the host/parasite phylogeny reconciliation problem. Math. Biosci. 149(2), 191–223 (1998)

[18] Chauve, C., El-Mabrouk, N.: New Perspectives on Gene Family Evolution: Losses in Reconciliation and a Link with Supertrees. In: Batzoglou, S. (ed.) RECOMB 2009. LNCS, vol. 5541, pp. 46–58. Springer, Heidelberg (2009)

[19] Doyon, J.-P., Scornavacca, C., Gorbunov, K.Y., Szöllősi, G.J., Ranwez, V., Berry, V.: An Efficient Algorithm for Gene/Species Trees Parsimonious Reconciliation with Losses, Duplications and Transfers. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 93–108. Springer, Heidelberg (2010)
[20] Eulenstein, O., Mirkin, B., Vingron, M.: Duplication-based measures of difference between gene and species trees. J. Comput. Biol. 5(1), 135–148 (1998)
[21] Fellows, M.R., Hallett, M.T., Stege, U.: On the Multiple Gene Duplication Problem. In: Chwa, K.-Y., Ibarra, O.H. (eds.) ISAAC 1998. LNCS, vol. 1533, pp. 347–356. Springer, Heidelberg (1998)
[22] Foley, D.H., Bryan, J.H., Yeates, D., Saul, A.: Evolution and systematics of anopheles: Insights from a molecular phylogeny of australasian mosquitoes. Mol. Phylogenet. Evol. 9(2), 262–275 (1998)
[23] Gogarten, M.B., Gogarten, J.P., Olendzenski, L. (eds.): Horizontal Gene Transfer - Genomes in Flux. Methods in Molecular Biology, vol. 532. Springer, Heidelberg (2009)
[24] Goodman, M., Czelusniak, J., Moore, G.W., Romero-Herrera, A.E., Matsuda, G.: Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. Syst. Zool. 28(2), 132–163 (1979)
[25] Górecki, P.: Reconciliation problems for duplication, loss and horizontal gene transfer. In: RECOMB, pp. 316–325 (2004)
[26] Górecki, P.: H-trees: a model of evolutionary scenarios with horizontal gene transfer. Fund. Inform. 103, 105–128 (2010)
[27] Górecki, P., Burleigh, J.G., Eulenstein, O.: Maximum likelihood models and algorithms for gene tree evolution with duplications and losses. BMC Bioinformatics 12(suppl. 1), S15 (2011)
[28] Górecki, P., Eulenstein, O.: A Linear Time Algorithm for Error-Corrected Reconciliation of Unrooted Gene Trees. In: Chen, J., Wang, J., Zelikovsky, A. (eds.) ISBRA 2011. LNCS, vol. 6674, pp. 148–159. Springer, Heidelberg (2011)
[29] Górecki, P., Tiuryn, J.: Dls-trees: A model of evolutionary scenarios. Theor. Comput. Sci. 359(1-3), 378–399 (2006)
[30] Górecki, P., Tiuryn, J.: Inferring phylogeny from whole genomes. Bioinformatics 23(2), e116–e222 (2007)
[31] Górecki, P., Tiuryn, J.: Urec: a system for unrooted reconciliation. Bioinformatics 23(4), 511–512 (2007)
[32] Guigó, R., Muchnik, I.B., Smith, T.F.: Reconstruction of ancient molecular phylogeny. Mol. Phylogenet. Evol. 6(2), 189–213 (1996)
[33] Hallett, M.T., Lagergren, J.: New algorithms for the duplication-loss model. In: RECOMB, pp. 138–146 (2000)
[34] Hallett, M.T., Lagergren, J.: Efficient algorithms for lateral gene transfer problems. In: RECOMB, pp. 149–156 (2001)
[35] Hallett, M.T., Lagergren, J., Tofigh, A.: Simultaneous identification of duplications and lateral transfers. In: RECOMB, pp. 347–356 (2004)
[36] Hill, T., Nordstrom, K., Thollesson, M., Safstrom, T., Vernersson, A., Fredriksson, R., Schioth, H.: SPRIT: Identifying horizontal gene transfer in rooted phylogenetic trees. BMC Evol. Biol. 10(1) (February 2010)
[37] Korochkina, S., Barreau, C., Pradel, G., Jeffery, E., Li, J., Natarajan, R., Shabanowitz, J., Hunt, D., Frevert, U., Vernick, K.D.: A mosquito-specific protein family includes candidate receptors for malaria sporozoite invasion of salivary glands. Cell Microbiol. 8, 163–175 (2006)
[38] Libeskind-Hadas, R., Charleston, M.A.: On the computational complexity of the reticulate cophylogeny reconstruction problem. J. Comput. Biol. 16(1), 105–117 (2009)

[39] Ma, B., Li, M., Zhang, L.: From gene trees to species trees. SIAM J. Comput. 30(3), 729–752 (2000)
[40] Mirkin, B., Muchnik, I.B., Smith, T.F.: A biologically consistent model for comparing molecular phylogenies. J. Comput. Biol. 2(4), 493–507 (1995)
[41] Nakhleh, L., Linder, C.R., Warnow, T., John, K.S.: Reconstructing reticulate evolution in species - theory and practice. In: Proc. of 8th Annual International Conference on Computational Molecular Biology, pp. 337–346 (2004)
[42] Ochiai, K., Yamanaka, T., Kimura, K., Sawada, O.: Inheritance of drug resistance (and its transfer) between Shigella strains and between Shigella and E. coli strains. Hihon Iji Shimpor, 1861 (1959) (in Japanese)
[43] Page, R.D.M.: Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. Syst. Biol. 43(1), 58–77 (1994)
[44] Page, R.D.M.: GeneTree: comparing gene and species phylogenies using reconciled trees. Bioinformatics 14(9), 819–820 (1998)
[45] Than, C., Ruths, D., Innan, H., Nakhleh, L.: Confounding factors in hgt detection: Statistical error, coalescent effects, and multiple solutions. J. Comput. Biol. 14, 517–535 (2007)
[46] Tofigh, A., Hallett, M.T., Lagergren, J.: Simultaneous identification of duplications and lateral gene transfers. IEEE/ACM Trans. Comput. Biology Bioinform. 8(2), 517–535 (2011)
[47] Chang, W.-C., Eulenstein, O.: Reconciling Gene Trees with Apparent Polytomies. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 235–244. Springer, Heidelberg (2006)
[48] Woolfit, M., Iturbe-Ormaetxe, I., McGraw, E.A., O'Neill, S.L.: An Ancient Horizontal Gene Transfer between Mosquito and the Endosymbiotic Bacterium Wolbachia pipientis. Mol. Biol. Evol. 26(2), 367–374 (2009)
[49] Zhang, L., Ng, Y.K., Wu, T., Zheng, Y.: Network model and efficient method for detecting relative duplications or horizontal gene transfers. In: Mandoiu, I.I., Miyano, S., Przytycka, T.M., Rajasekaran, S. (eds.) ICCABS, pp. 214–219. IEEE Computer Society (2011)

Capsules and Closures: A Small-Step Approach

Jean-Baptiste Jeannin

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
jeannin@cs.cornell.edu

Abstract. We present a side by side comparison of *Capsules* and *Closures*, including a proof of bisimilarity, using small-step semantics. A similar proof was presented in [8], using big-step semantics. However, while big-step semantics only allow to talk about final results of terminating computations, the use of small-step semantics allows to prove a stronger bisimilarity involving every step of the computation and thus also applicable to infinite computations.

1 Introduction

This paper compares the use of Capsules and Closures for a language with both higher-order and imperative features, and using small-step semantics. Capsules, introduced in [9], are a simple way of modeling the state of a computation for languages that are both imperative and functional. The state of a computation has been studied extensively [1,2,6,7,12–16,20,21]. However capsules intend to be as simple as possible, using only a single environment, while still capturing lexical scoping, variable assignment and recursion without heaps, stacks or combinators, and only using simple types.

The first versions of Lisp implemented *dynamic scoping*, which did not follow the semantics of the λ-calculus based on β-reduction. The language Scheme [22] fixed this by introducing *closures*, which allow to correctly model static scoping. The idea behind closures is to keep with each λ–abstraction the environment in which it was declared, thus forming a closure and allowing to execute the body of the λ-abstraction in the correct environment.

The language we introduce is both functional and imperative: it has higher-order functions, but every variable is mutable. This leads to interesting interactions and allows to go further than just enforcing lexical scoping. In particular, what do we expect the result of an expression like ($\mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0$) to be? Scheme (using set! for :=) and OCaml (using references) answer 2. Capsules give a rigorous mathematical definition that agrees and conservatively extends the scoping rules of the λ-calculus. Our semantics of closures also agrees with this definition, but this requires introducing a level of indirection, with both a stack of environments and a store, à la ML. Finally, recursive definitions are often implemented using some sort of backpatching; we build this directly

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 106–123, 2012.

into the definition of the language by defining $\mathsf{let\,rec}\ x = d\ \mathsf{in}\ e$ as a syntactic sugar for $\mathsf{let}\ x = a\ \mathsf{in}\ x := d; e$, where a is any expression of the appropriate type.

There is much previous work on reasoning about references and local state; see [11, 17–19]. State is typically modeled by some form of heap from which storage locations can be allocated and deallocated [7,15,20,21]. Others have used game semantics to reason about local state [4,5,10]. Mason and Talcott [12–14] and Felleisen and Hieb [6] present a semantics based on a heap and storage locations. A key difference is that Felleisen and Hieb's semantics is based on continuations. Finally, Moggi [16] proposed monads, which can be used to model state and are implemented in Haskell.

This paper is an improvement over [8] and borrows most of its structure, as we prove the same kind of result. Here we use small-step semantics instead of big-step semantics, which allows to prove bisimilar even infinite computations. The language and the notations are the same. The intended semantics of the language, both using capsules and closures, is the same as well, even though we have not formally proven the big-step and the small-step versions equivalent.

This paper is organized as follows. In Sect. 2, we formally introduce a programming language based on the λ-calculus containing both functional and imperative features. In Sect. 3, we describe two semantics for this language, one based on capsules and the other on closures. In Sect. 4, we show a very strong correspondence (Theorems 1 and 2, Corollary 1) between the two semantics, showing that every computation in the semantics of capsules is bisimilar to a computation in the semantics of closures, and vice-versa. We finish with a discussion in Sect. 5.

2 Syntax

2.1 Expressions

Expressions $\mathsf{Exp} = \{d, e, a, b, \ldots\}$ contain both functional and imperative features. There is an unlimited supply of *variables* $x, y, z, \ldots$ of all (simple) types, as well as *constants* $f, c, \ldots$ for primitive values. () is the only constant of type unit, and true and false are the only two constants of type bool. In the examples, $0, 1, 2, \ldots$ are predefined constants of type int. In addition, there are functional features.

- λ-abstraction $\lambda x.e$
- application $(d\ e)$,

imperative features

- assignment $x := e$
- composition $d; e$

- conditional — $\mathsf{if}\ b\ \mathsf{then}\ d\ \mathsf{else}\ e$
- while loop — $\mathsf{while}\ b\ \mathsf{do}\ e$,

and syntactic sugars

- $\mathsf{let}\ x = d\ \mathsf{in}\ e$ — $(\lambda x.e)\ d$
- $\mathsf{let\ rec}\ x = d\ \mathsf{in}\ e$ — $\mathsf{let}\ x = a\ \mathsf{in}\ x := d; e$

where a is any expression of the appropriate type.

Let Var be the set of variables, Const the set of constants, and λ-Abs the set of λ-abstractions. Given an expression e, let $\mathsf{FV}(e)$ denote the set of free variables of e. Given a partial function $h : \mathsf{Var} \rightharpoonup \mathsf{Var}$ such that $\mathsf{FV}(e) \subseteq \mathsf{dom}\, h$, let $h(e)$ be the expression e where every instance of a free variable $x \in \mathsf{FV}(e)$ has been replaced by the variable $h(x)$. As usual, given two partial functions g and h, $g \circ h$ denotes their composition such that for all x, $g \circ h(x) = g(h(x))$. Given a function h, we write $h[x/v]$ the function such that $h[x/v](y) = h(y)$ for $y \neq x$ and $h[x/v](x) = v$. Given an expression e, we write $e[x/y]$ the expression e where all free occurrences of x have been replaced by y.

Throughout the paper, we focus on the features directly involving variables: variable calls x, λ-abstractions $\lambda x.e$, applications $(d\ e)$ where d reduces to a λ-abstraction, and assignment $x := e$. Most differences between capsules and closures arise using these features.

2.2 Types

Types $\alpha, \beta, \ldots$ are built inductively from an unspecified family of base types, including at least unit and bool, and a type constructor $\to$ such that functions with input type α and return type β have type $\alpha \to \beta$. All constants c of the language have a type $\mathsf{type}(c)$; by convention, we use c for a constant of a base type and f for a constant of a functional type. We follow [23] in assuming that each variable x is associated with a unique type $\mathsf{type}(x)$, that could for example be built into the variable name. Γ is a type environment, a partial function $\mathsf{Var} \rightharpoonup \mathsf{Type}$. As is standard, we write $\Gamma, x : \alpha$ for the typing environment Γ where x has been bound or rebound to α. The typing rules are standard:

$$\Gamma \vdash c : \alpha \text{ if } \mathsf{type}(c) = \alpha \qquad \Gamma, x : \alpha \vdash x : \alpha \qquad \frac{\mathsf{type}(x) = \alpha \quad \Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash \lambda x.e : \alpha \to \beta}$$

$$\frac{\Gamma \vdash d : \alpha \to \beta \quad \Gamma \vdash e : \alpha}{\Gamma \vdash (d\ e) : \beta} \qquad \frac{\Gamma \vdash x : \alpha \quad \Gamma \vdash e : \alpha}{\Gamma \vdash x := e : \mathsf{unit}} \qquad \frac{\Gamma \vdash d : \mathsf{unit} \quad \Gamma \vdash e : \alpha}{\Gamma \vdash d; e : \alpha}$$

$$\frac{\Gamma \vdash b : \mathsf{bool} \quad \Gamma \vdash d : \alpha \quad \Gamma \vdash e : \alpha}{\Gamma \vdash \mathsf{if}\ b\ \mathsf{then}\ d\ \mathsf{else}\ e : \alpha} \qquad \frac{\Gamma \vdash b : \mathsf{bool} \quad \Gamma \vdash e : \mathsf{unit}}{\Gamma \vdash \mathsf{while}\ b\ \mathsf{do}\ e : \mathsf{unit}}$$

3 Semantics

We present two different semantics that have a strong correspondence:

- The semantics on *capsules* is a simplified version of the semantics on closure structures introduced in [3]. It has previously been described in [9];
- The semantics on *closures* is the semantics usually used and taught for functional languages. A level of indirection for variables has been added to support imperative features, *à la* ML. Developing a small-step semantics for closures ended up being a little bit more complicated than expected.

From now on all the expressions we consider are supposed well-typed with the rules of Sect. 2.2.

3.1 Capsules

Definitions. An *irreducible term* is either a constant or a λ-abstraction. A *capsule environment* γ is a partial function from variables to irreducible terms, such that

$$\forall x \in \mathsf{dom}\,\gamma,\ \mathsf{FV}(\gamma(x)) \subseteq \mathsf{dom}\,\gamma.$$

Let $i, j, k, \ldots$ denote irreducible terms and $\gamma, \delta, \zeta, \eta, \ldots$ capsule environments. Let $\mathsf{Irred} = \mathsf{Const} + \lambda\text{-}\mathsf{Abs}$ be the set of irreducible terms. Thus we have:

$$\gamma : \mathsf{Var} \rightharpoonup \mathsf{Irred} \qquad\qquad \mathsf{Irred} = \mathsf{Const} + \lambda\text{-}\mathsf{Abs}$$

Semantics. A *capsule* is a pair $\langle e, \gamma \rangle$, such that $\mathsf{FV}(e) \subseteq \mathsf{dom}\,\gamma$.

Let us define a small step semantics where the operator $\rightarrow_{\mathsf{ca}}$ relates capsules. The semantics of features directly involving variables is given by:

$$\langle x, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle \gamma(x), \gamma \rangle \qquad \langle y := i, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle (), \gamma[y/i] \rangle$$
$$\langle (\lambda x.b)\ i, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle b[x/y], \gamma[y/i] \rangle \quad (y \text{ fresh})$$

and the remaining semantics is:

$$\langle f\ c, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle f(c), \gamma \rangle \qquad \langle (); e, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle e, \gamma \rangle$$
$$\langle \mathsf{if\ true\ then}\ d\ \mathsf{else}\ e, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle d, \gamma \rangle \qquad \langle \mathsf{if\ false\ then}\ d\ \mathsf{else}\ e, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle e, \gamma \rangle$$
$$\langle \mathsf{while}\ b\ \mathsf{do}\ e, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle \mathsf{if}\ b\ \mathsf{then}\ (e; \mathsf{while}\ b\ \mathsf{do}\ e)\ \mathsf{else}\ (), \gamma \rangle$$

Evaluation contexts C are defined by:

$$C ::= \quad [\cdot] \quad | \quad C\ e \quad | \quad i\ C \quad | \quad x := C \quad | \quad C; e \quad | \quad \mathsf{if}\ C\ \mathsf{then}\ d\ \mathsf{else}\ e$$

where each evaluation context $C[\,\cdot\,]$ generates a rule:

$$\frac{\langle d,\, \gamma\rangle \to_{\mathsf{ca}} \langle e,\, \delta\rangle}{\langle C[d],\, \gamma\rangle \to_{\mathsf{ca}} \langle C[e],\, \delta\rangle}$$

The well-typed final capsules, i.e. capsules that cannot take a small step, are exactly the capsules $\langle i,\, \gamma\rangle$ for any irreducible term i.

As usual, we introduce $\to_{\mathsf{ca}}^*$ as the reflexive transitive closure of $\to_{\mathsf{ca}}$.

Examples. The following examples show that lexical scoping and recursion are handled.

Example 1. $(\mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0) \to_{\mathsf{ca}}^* 1$.

Proof

$$\begin{aligned}
& \langle\ \mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, && [\,]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ \mathsf{let}\ f = \lambda y.x'\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, && [x' = 1]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ \mathsf{let}\ x = 2\ \mathsf{in}\ f'\ 0, && [x' = 1,\ f' = \lambda y.x']\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ f'\ 0, && [x' = 1,\ f' = \lambda y.x',\ x'' = 2]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ (\lambda y.x')\ 0, && [x' = 1,\ f' = \lambda y.x',\ x'' = 2]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ x', && [x' = 1,\ f' = \lambda y.x',\ x'' = 2,\ y' = 0]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ 1, && [x' = 1,\ f' = \lambda y.x',\ x'' = 2,\ y' = 0]\ \rangle
\end{aligned}$$

Example 2. $(\mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0) \to_{\mathsf{ca}}^* 2$.

Proof

$$\begin{aligned}
& \langle\ \mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0, && [\,]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ \mathsf{let}\ f = \lambda y.x'\ \mathsf{in}\ x' := 2; f\ 0, && [x' = 1]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ x' := 2; f'\ 0, && [x' = 1,\ f' = \lambda y.x']\ \rangle \\
\to_{\mathsf{ca}}^*\ & \langle\ f'\ 0, && [x' = 2,\ f' = \lambda y.x']\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ (\lambda y.x')\ 0, && [x' = 2,\ f' = \lambda y.x']\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ x', && [x' = 2,\ f' = \lambda y.x',\ y' = 0]\ \rangle \\
\to_{\mathsf{ca}}\ & \langle\ 2, && [x' = 2,\ f' = \lambda y.x',\ y' = 0]\ \rangle
\end{aligned}$$

Example 3. $(\mathsf{let\,rec}\ f = \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n\ \mathsf{in}\ f\ 3) \to_{\mathsf{ca}}^* 6$.

Proof. In this example e stands for $\lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n$.

$$
\begin{aligned}
& \langle\ \mathsf{let\,rec}\ f = \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n\ \mathsf{in}\ f\ 3, && [\,]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ f\ 3, && [f = \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ \mathsf{if}\ n_1 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n_1-1) \times n_1, && [f = e,\ n_1 = 3]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (f\ 2) \times n_1, && [f = e,\ n_1 = 3]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (\mathsf{if}\ n_2 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n_2 \times f(n_2-1)) \times n_1, && [f = e,\ n_1 = 3,\ n_2 = 2]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (f\ 1) \times n_2 \times n_1, && [f = e,\ n_1 = 3,\ n_2 = 2]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (\mathsf{if}\ n_3 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n_3 \times f(n_3-1)) \times n_2 \times n_1, \\
& && [f = e,\ n_1 = 3,\ n_2 = 2,\ n_3 = 1]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (f\ 0) \times n_3 \times n_2 \times n_1, && [f = e,\ n_1 = 3,\ n_2 = 2,\ n_3 = 3]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ (\mathsf{if}\ n_4 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n_4 \times f(n_4-1)) \times n_3 \times n_2 \times n_1, \\
& && [f = e,\ n_1 = 3,\ n_2 = 2,\ n_3 = 1,\ n_4 = 0]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ 1 \times n_3 \times n_2 \times n_1, && [f = e,\ n_1 = 3,\ n_2 = 2,\ n_3 = 1,\ n_4 = 0]\ \rangle \\
\to^*_{\mathsf{ca}}\ & \langle\ 6, && [f = e,\ n_1 = 3,\ n_2 = 2,\ n_3 = 1,\ n_4 = 0]\ \rangle
\end{aligned}
$$

3.2 Closures

Definitions. Closures were introduced in the language Scheme [22]. We present a version of them using a level of indirection, allowing us to handle mutable variables.

There is an unlimited number of locations $\ell, \ell_1, \ell_2 \ldots$; locations can be thought of as addresses in memory. An *environment* is a partial function from variables to locations. A *closure* is defined as a pair $\{\lambda x.e,\ \sigma\}$ such that $\mathsf{FV}(\lambda x.e) \subseteq \mathsf{dom}\,\sigma$, where $\lambda x.e$ is a λ-abstraction and σ is an environment that is used to interpret the free variables of $\lambda x.e$. A *value* is either a constant or a closure. Values for closures play the same role as irreducible terms for capsules. A *store* (or *memory*) is a partial function from locations to values.

Let $u, v, w, \ldots$ denote values, $\sigma, \tau, \ldots$ environments and $\mu, \nu, \xi, \chi, \ldots$ stores. Let Val be the set of values, Loc the set of locations and Cl the set of closures. Thus we have:

$$\sigma : \mathsf{Var} \rightharpoonup \mathsf{Loc} \qquad \mu : \mathsf{Loc} \rightharpoonup \mathsf{Val} \qquad \mathsf{Val} = \mathsf{Const} + \mathsf{Cl}$$

The interaction of small-step semantics and closures leads to using stacks of environments: when entering the body of a function, the environment coming with its closure is pushed; and when leaving this body, it is popped. Let $\Sigma, \Pi, \ldots$ denote stacks of environments. Let us write $[\sigma]$ the stack of environments containing only the element σ, as to not confuse it with the single environment σ. $\sigma :: \tau$ represents the stack containing σ at the top of the stack and τ at its bottom. $\sigma :: \Sigma$ represents the stack Σ with σ added on top of it; and $\Sigma :: \sigma$ represents Σ with σ added at its bottom. We define $\mathsf{hd}(\Sigma) = \sigma$ and $\mathsf{tl}(\Sigma) = \Pi$ whenever $\Sigma = \sigma :: \Pi$.

To define a small-step semantics of closures, we need to represent all the different shapes an expression can take throughout computation, until it becomes a value. State expressions $\mathsf{StExp} = \{s, t, \ldots\}$ allow to do this. Of course, expressions and values are state expressions, but some state expressions are neither.

$$\mathsf{Exp} \subseteq \mathsf{StExp} \qquad\qquad \mathsf{Val} \subseteq \mathsf{StExp}$$

A *state expression* can be:

- an expression e; this includes constants c;
- a closure $\{\lambda x.a,\, \sigma\}$;
- an evaluation expression followed by a pop indicator $\square$, as in $s\ \square$; when entering the body of a function, a new environment needs to be pushed on the stack of environments; this environment needs to be popped when leaving the body of the function; one way to know when this happens is to keep track of the end of the body with $\square$;
- an evaluation expression applied to an expression, $s\ e$;
- a value applied to an evaluation expression, $v\ s$;
- an assignment, $x := s$;
- a composition $s; e$;
- an if statement $\mathsf{if}\ s\ \mathsf{then}\ d\ \mathsf{else}\ e$.

We extend the notion of free variables to an evaluation expression in a natural, syntactic way: for a well-formed closure $\{\lambda x.a,\, \sigma\}$ with $\mathsf{FV}(\lambda x.a) \subseteq \mathsf{dom}\,\sigma$, $\mathsf{FV}(\{\lambda x.a,\, \sigma\})$ is the empty set; and $\mathsf{FV}(s\ \square) = \mathsf{FV}(s)$.

Stacks of environments, along with the introduction of closures and of the pop indicator $\square$, are a convenient way to model in which environment each variable should be looked up. Intuitively, any free variable in a λ-abstraction of a closure should be interpreted in the environment coming with this closure. Because of the definition of state expressions, the pop indicators all are inside each other. The variables inside the deepest pop indicator are interpreted in the environment on top the stack; the variables inside the second deepest but outside the deepest pop indicator are interpreted in the second environment from the top of the stack, and so on. The variables outside of any pop indicator are interpreted in the environment at the bottom of the stack. A precise account of this idea will be given in Sect. 4.1 with the definition of $h \circ \Sigma$.

Semantics. A *state* is a triple $(s,\, \Sigma,\, \mu)$.

$$\mathsf{FV}(s) \subseteq \mathsf{dom}\,(\mathsf{hd}\,\Sigma) \qquad\qquad \forall \sigma \in \Sigma, \mathsf{codom}\,\sigma \subseteq \mathsf{dom}\,\mu$$
$$\forall \{\lambda x.a,\, \tau\} \in \mathsf{codom}\,\mu,\ \mathsf{FV}(\lambda x.a) \subseteq \mathsf{dom}\,\tau \wedge \mathsf{codom}\,\tau \subseteq \mathsf{dom}\,\mu$$

When evaluating an expression e, we start with the initial state $(e,\, [\sigma],\, \mu)$ where σ and μ are both empty mappings. Let us define a small step semantics where the operator $\to_{\mathsf{cl}}$ relates valid states to valid results. The semantics of features directly involving variables is given by:

$$(x,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (\mu(\sigma(x)),\, [\sigma],\, \mu) \qquad (\lambda x.a,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (\{\lambda x.a,\, \sigma\},\, [\sigma],\, \mu)$$
$$(\{\lambda x.a, \sigma\}\ v,\, [\tau],\, \mu) \to_{\mathsf{cl}} (a\ \Box,\, \sigma[x/\ell] :: \tau,\, \mu[\ell/v]) \quad (\ell \text{ fresh})$$
$$(v\ \Box,\, \sigma :: \tau,\, \mu) \to_{\mathsf{cl}} (v,\, [\tau],\, \mu) \qquad (x := v,\, [\sigma],\, \mu) \to_{\mathsf{cl}} ((),\, [\sigma],\, \mu[\sigma(x)/v])$$

and the remaining semantics is:

$$(f\ c,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (f(c),\, [\sigma],\, \mu) \qquad ((); e,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (e,\, [\sigma],\, \mu)$$
$$(\mathsf{if\ true\ then}\ d\ \mathsf{else}\ e,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (d,\, [\sigma],\, \mu)$$
$$(\mathsf{if\ false\ then}\ d\ \mathsf{else}\ e,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (e,\, [\sigma],\, \mu)$$
$$(\mathsf{while}\ b\ \mathsf{do}\ e,\, [\sigma],\, \mu) \to_{\mathsf{cl}} (\mathsf{if}\ b\ \mathsf{then}\ (e; \mathsf{while}\ b\ \mathsf{do}\ e)\ \mathsf{else}\ (),\, [\sigma],\, \mu)$$

Evaluation contexts C are defined by:

$$C ::= \quad [\cdot] \quad | \quad C\ e \quad | \quad v\ C \quad | \quad x := C \quad | \quad C; e \quad | \quad \mathsf{if}\ C\ \mathsf{then}\ d\ \mathsf{else}\ e$$

where each evaluation context $C[\,\cdot\,]$ generates a rule:

$$\frac{(s,\, \Sigma,\, \mu) \to_{\mathsf{cl}} (t,\, \Pi,\, \nu)}{(C[s],\, \Sigma,\, \mu) \to_{\mathsf{cl}} (C[t],\, \Pi,\, \nu)}$$

One more rule is needed to be able to evaluate under a pop indicator $\Box$:

$$\frac{(s,\, \Sigma,\, \mu) \to_{\mathsf{cl}} (t,\, \Pi,\, \nu)}{(s\ \Box,\, \Sigma :: \sigma,\, \mu) \to_{\mathsf{cl}} (t\ \Box,\, \Pi :: \sigma,\, \nu)}$$

Note the similarity between the last two rules, including the definition of evaluation contexts, and the inductive definition of state environments. This is not by chance: the innermost state expression, if not a value, is always the one which will be evaluated next.

The final states, i.e., the states that cannot take a small step, are the (v, Σ, μ) for any value v.

As usual, we introduce $\to_{\mathsf{cl}}^*$ as the reflexive transitive closure of $\to_{\mathsf{cl}}$.

Properties. Some properties of this semantics can be easily proved:

- In an evaluation expression s, all the pop indicators $\Box$ are inside each other; the deepest one is inside all the others, and so on.
- If starting from an initial state, the number of elements on the environment stack Σ is always one more than the number of pop indicators $\Box$ in s.
- if $\Sigma \to_{\mathsf{cl}} \Pi$ then either $\Sigma = \Pi$ or $\Sigma = \sigma :: \Pi$ or $\sigma :: \Sigma = \Pi$ for some σ.

Examples. To illustrate the above semantics, we now show the evaluation of the examples of section 3.1 using closures.

Example 4. $(\mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0) \to_{\mathsf{cl}}^{*} 1$.

Proof

$$
\begin{aligned}
& \langle\ \mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, && [\,], [\,]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \{\lambda x.\mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, [\,]\}\ 1, && [\,], [\,]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0\ \square, && [x = \ell_1] :: [\,], [\ell_1 = 1]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \{\lambda f.\mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, [x = \ell_1]\}\ \lambda y.x\ \square, && [x = \ell_1] :: [\,], [\ell_1 = 1]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \{\lambda f.\mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0, [x = \ell_1]\}\ \{\lambda y.x, [x = \ell_1]\}\ \square, && [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \mathsf{let}\ x = 2\ \mathsf{in}\ f\ 0\ \square\ \square, && [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \{\lambda x.f\ 0, [x = \ell_1, f = \ell_2]\}\ 2\ \square\ \square, && [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ f\ 0\ \square\ \square\ \square, && [f = \ell_2, x = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ \{\lambda y.x, [x = \ell_1]\}\ 0\ \square\ \square\ \square, && \\
& && [f = \ell_2, x = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ x\ \square\ \square\ \square\ \square, && \\
& && [x = \ell_1, y = \ell_4] :: [f = \ell_2, x = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ 1\ \square\ \square\ \square\ \square, && \\
& && [x = \ell_1, y = \ell_4] :: [f = \ell_2, x = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ 1\ \square\ \square\ \square, && [f = \ell_2, x = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ 1\ \square\ \square, && [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
& && [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ 1\ \square, && [x = \ell_1] :: [\,], [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ 1, && [\,], [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 2, \ell_4 = 0]\ \rangle
\end{aligned}
$$

Example 5. $(\mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0) \rightarrow_{\mathsf{cl}}^{*} 2$.

Proof

$$
\begin{array}{lll}
 & \langle\ \mathsf{let}\ x = 1\ \mathsf{in}\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0, & [\,], [\,]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ \{\lambda x.\mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0, [\,]\}\ 1, & [\,], [\,]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ \mathsf{let}\ f = \lambda y.x\ \mathsf{in}\ x := 2; f\ 0\ \square, & [x = \ell_1] :: [\,], [\ell_1 = 1]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ \{\lambda f.x := 2; f\ 0, [x = \ell_1]\}\ \lambda y.x\ \square, & [x = \ell_1] :: [\,], [\ell_1 = 1]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ \{\lambda f.x := 2; f\ 0, [x = \ell_1]\}\ \{\lambda y.x, [x = \ell_1]\}\ \square, & [x = \ell_1] :: [\,], [\ell_1 = 1]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ x := 2; f\ 0\ \square\ \square, & [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
 & & [\ell_1 = 1, \ell_2 = \{\lambda y.x, [x = \ell_1]\}]\ \rangle \\
\rightarrow_{\mathsf{cl}}^{*} & \langle\ f\ 0\ \square\ \square, [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ \{\lambda y.x, [x = \ell_1]\}\ 0\ \square\ \square, & [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
 & & [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ x\ \square\ \square\ \square, & [x = \ell_1, y = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
 & & [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 0]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ 2\ \square\ \square\ \square, & [x = \ell_1, y = \ell_3] :: [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
 & & [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 0]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ 2\ \square\ \square, & [x = \ell_1, f = \ell_2] :: [x = \ell_1] :: [\,], \\
 & & [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 0]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ 2\ \square, & [x = \ell_1] :: [\,], [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 0]\ \rangle \\
\rightarrow_{\mathsf{cl}} & \langle\ 2, & [\,], [\ell_1 = 2, \ell_2 = \{\lambda y.x, [x = \ell_1]\}, \ell_3 = 0]\ \rangle
\end{array}
$$

Example 6. $(\mathsf{let\,rec}\ f = \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n\ \mathsf{in}\ f\ 3) \rightarrow_{\mathsf{cl}}^{*} 6$.

Proof. This example is particularly interesting as it shows how nested $\square$ allow to interpret the same variable in different scopes. In all the example e stands for $\{\lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n, [f = \ell_1]\}$, and d stands for $\{\lambda n.n, [\,]\}$, a dummy value used when creating the recursive function f.

$$
\begin{aligned}
& \langle\ \mathsf{let\,rec}\ f = \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n\ \mathsf{in}\ f\ 3, && [\,], [\,]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ f := \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n; f\ 3\ \square, && [f = \ell_1] :: [\,], [\ell_1 = d]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ f\ 3\ \square, && [f = \ell_1] :: [\,], [\ell_1 = e]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ \mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ f(n-1) \times n\ \square\ \square, && [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ (f\ 2) \times n\ \square\ \square, && [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ (\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f(n-1)\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ ((f\ 1) \times n\ \square) \times n\ \square\ \square, && [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ ((\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f(n-1)\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ (((f\ 0) \times n\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ (((\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f(n-1)\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_5] :: [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: \\
& && [f = \ell_1] :: [\,], [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ (((\mathsf{if}\ 0 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f(n-1)\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_5] :: [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: \\
& && [f = \ell_1] :: [\,], [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ (((1\ \square) \times n\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_5] :: [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: \\
& && [f = \ell_1] :: [\,], [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to_{\mathsf{cl}}\ & \langle\ ((1 \times n\ \square) \times n\ \square) \times n\ \square\ \square, \\
& && [f = \ell_1, n = \ell_4] :: [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ ((1 \times 1) \times n\ \square) \times n\ \square\ \square, [f = \ell_1, n = \ell_3] :: [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ ((1 \times 1) \times 2) \times n\ \square\ \square, && [f = \ell_1, n = \ell_2] :: [f = \ell_1] :: [\,], \\
& && [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle \\
\to^*_{\mathsf{cl}}\ & \langle\ 6, && [\,], [\ell_1 = e, \ell_2 = 3, \ell_3 = 2, \ell_4 = 1, \ell_5 = 0]\ \rangle
\end{aligned}
$$

4 Equivalence of the Small-Step Semantics

4.1 Definitions

There is a very strong correspondence between the semantics of closures and capsules. To give a precise account of this correspondence, we introduce an injective partial function $h : \mathsf{Loc} \rightharpoonup \mathsf{Var}$ with which we define three relations. Each relation is between an element of the semantics of closures and an element of the semantics of capsules that play similar roles:

- $v \xrightarrow{h} i$ between values and irreducible terms;
- $\mu \xrightarrow{h} \gamma$ between stores and capsule environments;
- $(s, \Sigma, \mu) \overset{h}{\sim} \langle e, \gamma \rangle$ between states and capsules;

One thing to notice is that nothing in the semantics of capsules plays the same role as the stack of environments Σ in the semantics of closures: capsule environments γ relate to memories μ, and the stack of environments Σ has been simplified. Let us now give precise definitions of those relations.

Definition 1. *Given a value v and an irreducible term i, we say that* h transforms v into i, *where h is an injective map $h : \mathsf{Loc} \rightharpoonup \mathsf{Var}$, and we write $v \xrightarrow{h} i$, if and only if:*

- $v = i$ *when* $v \in \mathsf{Const}$, *or*
- $\mathsf{codom}\,\tau \subseteq \mathsf{dom}\,h$ *and* $(h \circ \tau)(\lambda x.a) = i$ *when* $v = \{\lambda x.a,\, \tau\} \in \mathsf{Cl}$

Definition 2. *Given a store μ and a capsule environment γ, we say that h* transforms μ into γ, *where h is an injective map $h : \mathsf{Loc} \rightharpoonup \mathsf{Var}$, and we write $\mu \xrightarrow{h} \gamma$, if and only if:*

$$\mathsf{dom}\,h = \mathsf{dom}\,\mu \qquad h(\mathsf{dom}\,\mu) = \mathsf{dom}\,\gamma$$
$$\forall \ell \in \mathsf{dom}\,\mu,\ \mu(\ell) \xrightarrow{h} \gamma(h(\ell))$$

We are now ready to give a precise account of the interpretation of variables in state environments, as described in Sect. 3.2. Given a map $h : \mathsf{Loc} \rightharpoonup \mathsf{Var}$ and a stack of environments Σ, let us inductively define the operator $h \circ \Sigma : \mathsf{StExp} \to \mathsf{Exp}$ as:

$$\begin{aligned}
h \circ (\Sigma :: \sigma)(e) &= h \circ \sigma(e) \\
h \circ \Sigma(\{\lambda x.a,\, \sigma\}) &= \sigma(\lambda x.a) \\
h \circ (\Sigma :: \sigma)(s\ \square) &= h \circ \Sigma(s) \\
h \circ \Sigma(s\ e) &= (h \circ \Sigma(s))\ (h \circ \Sigma(e)) \\
h \circ \Sigma(v\ s) &= (h \circ \Sigma(v))\ (h \circ \Sigma(s)) \\
h \circ (\Sigma :: \sigma)(x := s) &= (h \circ \sigma(x)) := h \circ (\Sigma :: \sigma)(s) \\
h \circ \Sigma(s; e) &= h \circ \Sigma(s); h \circ \Sigma(e) \\
h \circ \Sigma(\mathsf{if}\ s\ \mathsf{then}\ d\ \mathsf{else}\ e) &= \mathsf{if}\ h \circ \Sigma(s)\ \mathsf{then}\ h \circ \Sigma(d)\ \mathsf{else}\ h \circ \Sigma(e)
\end{aligned}$$

Definition 3. *Given a state* (s, Σ, μ) *and a capsule* $\langle e, \gamma \rangle$*, both valid, we say that they are* bisimilar under h*, where* h *is an injective map* $h : \mathsf{Loc} \rightharpoonup \mathsf{Var}$*, and we write* $(s, \Sigma, \mu) \overset{h}{\sim} \langle e, \gamma \rangle$*, if and only if*

$$h \circ \Sigma(s) = e \qquad\qquad \mu \xrightarrow{h} \gamma$$

4.2 Soundness of Capsules with Respect to Closures

Now that we know how to relate each element of both semantics, Theorem 1 shows that any derivation using closures mirrors zero or more derivation steps using capsules, and Theorem 2 shows that any derivation step using capsules mirrors zero or more derivation steps using closures. Combined, they give rise to Corollary 1, which shows that any derivation using capsules is mirrored by a derivation using closures, and vice-versa.

Theorem 1. *If* $(s, \Sigma, \mu) \overset{h}{\sim} \langle d, \gamma \rangle$ *and* $(s, \Sigma, \mu) \to_{\mathsf{cl}} (t, \Pi, \nu)$*, then there exists* e, δ *such that*

$$\langle d, \gamma \rangle \to^*_{\mathsf{ca}} \langle e, \delta \rangle \qquad\qquad (t, \Pi, \nu) \overset{g}{\sim} \langle e, \delta \rangle$$

where g *is an extension of* h*, i.e.,* $\mathsf{dom}\, h \subseteq \mathsf{dom}\, g$ *and* h *and* g *agree on* $\mathsf{dom}\, h$*.*

Theorem 2. *If* $(s, \Sigma, \mu) \overset{h}{\sim} \langle d, \gamma \rangle$ *and* $\langle d, \gamma \rangle \to_{\mathsf{ca}} \langle e, \delta \rangle$*, then there exists* t, Π, ν *such that*

$$(s, \Sigma, \mu) \to^*_{\mathsf{cl}} (t, \Pi, \nu) \qquad\qquad (t, \Pi, \nu) \overset{g}{\sim} \langle e, \delta \rangle$$

where g *is an extension of* h*, i.e.,* $\mathsf{dom}\, h \subseteq \mathsf{dom}\, g$ *and* h *and* g *agree on* $\mathsf{dom}\, h$*.*

Corollary 1. *If* $(s, \Sigma, \mu) \overset{h}{\sim} \langle d, \gamma \rangle$ *then*

- *if* $(s, \Sigma, \mu) \to^*_{\mathsf{cl}} (t, \Pi, \nu)$ *then there exists* e, δ *such that*

$$\langle d, \gamma \rangle \to^*_{\mathsf{ca}} \langle e, \delta \rangle \qquad\qquad (t, \Pi, \nu) \overset{g}{\sim} \langle e, \delta \rangle$$

- *if* $\langle d, \gamma \rangle \to^*_{\mathsf{ca}} \langle e, \delta \rangle$ *then there exists* t, Π, ν *such that*

$$(s, \Sigma, \mu) \to^*_{\mathsf{cl}} (t, \Pi, \nu) \qquad\qquad (t, \Pi, \nu) \overset{g}{\sim} \langle e, \delta \rangle$$

where g *is an extension of* h*, i.e.,* $\mathsf{dom}\, h \subseteq \mathsf{dom}\, g$ *and* h *and* g *agree on* $\mathsf{dom}\, h$*.*

Proof of Corollary 1. We use standard arguments on weak bisimilarity. The first part is proved by recurrence on the number of steps of the derivation of $(s, \Sigma, \mu) \to^*_{\mathsf{cl}} (t, \Pi, \nu)$ and application of Theorem 1. Similarly, the second part is by recurrence on the number of steps of $\langle d, \gamma \rangle \to^*_{\mathsf{ca}} \langle e, \delta \rangle$ and application of Theorem 2. □

Proof of Theorem 1. We proceed by induction on the derivation of $(s, \Sigma, \mu) \to_{\mathsf{cl}} (t, \Pi, \nu)$. In the interest of space, we only show the most interesting cases of the induction in the main text: variable call x, λ-abstraction $\lambda x.e$, function

application of a closure $\{\lambda x.a,\, \sigma\}\ v$, popping from the environment stack $v\ \square$, variable assignment $x := e$, contexts $C[s]$ and computing with the pop indicator $s\ \square$. The other cases, function application of a constant function $f\ c$, composition $d; e$, if conditional $\mathsf{if}\ b\ \mathsf{then}\ d\ \mathsf{else}\ e$ and while loop $\mathsf{while}\ b\ \mathsf{do}\ e$, are straightforward inductive arguments.

Variable Call. If $s = x$ for some variable x and $\Sigma = [\sigma]$ then $d = (h \circ \sigma)(s) = y$ with y the variable such that $y = (h \circ \sigma)(x)$.

By definition of $\to_{\mathsf{cl}}$, $(t,\, \Pi,\, \nu) = (\mu(\sigma(x)),\, [\sigma],\, \mu)$, and by definition of $\to_{\mathsf{ca}}$, $\langle d,\, \gamma\rangle = \langle y,\, \gamma\rangle \to_{\mathsf{ca}} \langle \gamma(y),\, \gamma\rangle$. Moreover $\mu \overset{h}{\to} \gamma$, therefore by definition of $\overset{h}{\to}$, $\mu(\sigma(x)) \overset{h}{\to} \gamma(h(\sigma(x))) = \gamma(y)$. Therefore, with $g = h$, $(t,\, \Pi,\, \nu) = (\mu(\sigma(x)),\, [\sigma],\,)\mu \overset{g}{\sim} \langle \gamma(y),\, \gamma\rangle$ which completes this case.

λ-Abstraction. If $s = \lambda x.a$ and $\Sigma = [\sigma]$, then $d = (h \circ \sigma)(\lambda x.a)$ which is a term α-equivalent to s, so $d = \lambda x.b$ for some b. Indeed, the variable x does not change from s to d since only the free variables of s are affected by $h \circ \sigma$.

By definition of $\to_{\mathsf{cl}}$, $(t,\, \Pi,\, \nu) = (\{\lambda x.a,\, \sigma\},\, [\sigma],\, \mu)$, and by reflexivity of $\to^*_{\mathsf{ca}}$, $\langle d,\, \gamma\rangle = \langle \lambda x.b,\, \gamma\rangle \to^*_{\mathsf{ca}} \langle \lambda x.b,\, \gamma\rangle$. But $\mathsf{codom}\,\sigma \subseteq \mathsf{dom}\,h$ and $\lambda x.b = (h \circ \sigma)(\lambda x.a)$, therefore $\{\lambda x.a,\, \sigma\} \overset{h}{\to} \lambda x.b$. Moreover we know $\mu \overset{h}{\to} \gamma$ and with $g = h$, we get $(\{\lambda x.a,\, \sigma\},\, [\sigma],\, \mu) \overset{g}{\sim} \langle \lambda x.b,\, \gamma\rangle$ which completes this case.

Function Application of a Closure. If $s = \{\lambda x.a,\, \sigma\}\ v$ and $\Sigma = [\tau]$, then $(h \circ \Sigma)(\{\lambda x.a,\, \sigma\}) = (h \circ \sigma)(\lambda x.a)$ is a $\lambda x.b$ for some expression b, and $(h \circ \Sigma)(v)$ is some irreducible term i. Since $d = (h \circ \Sigma)(s)$, $d = (\lambda x.b)\ i$.

By definition of $\to_{\mathsf{cl}}$, $(t,\, \Pi,\, \nu) = (a\ \square,\, \sigma[x/\ell] :: \tau,\, \mu[\ell/v])$ with ℓ fresh, and by definition of $\to_{\mathsf{ca}}$, $\langle d,\, \gamma\rangle \to_{\mathsf{ca}} \langle b[x/y],\, \gamma[y/i]\rangle$, with y fresh. Let $g : \mathsf{Loc} \rightharpoonup \mathsf{Var}$ such that:

$$\begin{aligned} g : \mathsf{dom}\,h \cup \{\ell\} &\to \mathsf{codom}\,g \cup \{y\} \\ \ell_h \in \mathsf{dom}\,h &\mapsto h(\ell_h) \\ \ell &\mapsto y \end{aligned}$$

Lemma 1. $(a,\, [\sigma[x/\ell]],\, \mu[\ell/v]) \overset{g}{\sim} \langle b[x/y],\, \gamma[y/i]\rangle$.

Proof. First of all, $\lambda x.b = (h \circ \sigma)(\lambda x.a)$, g is an extension of h and $\mathsf{FV}(\lambda x.a) \subseteq \mathsf{dom}\,h$, therefore $\lambda x.b = (g \circ \sigma)(\lambda x.a)$. Now $b[x/y] = ((g \circ \sigma)[x/y])(a) = (g \circ \sigma[x/\ell])(\lambda x.a)$ since $g(\ell) = y$.

We further need to argue that $\mu[\ell/v] \overset{g}{\to} \gamma[y/i]$. We already know that $\mathsf{dom}\,g = \mathsf{dom}\,h \cup \{\ell\} = \mathsf{dom}\,\mu \cup \{\ell\} = \mathsf{dom}\,\mu[\ell/v]$, and $g(\mathsf{dom}\,\mu[\ell/v]) = \mathsf{codom}\,h \cup \{y\} = \mathsf{dom}\,\gamma[y/i]$. Let $\ell' \in \mathsf{dom}\,\mu[\ell/v]$. If $\ell' \in \mathsf{dom}\,\mu$, then $\mu[\ell/v](\ell') = \mu(\ell') \overset{h}{\to} \gamma(g(\ell')) = \gamma[y/i](g(\ell'))$ by injectivity of g, therefore $\mu[\ell/v](\ell') \overset{g}{\to} \gamma[y/i](g(\ell'))$. Otherwise, $\ell' = \ell$ and then $\mu[\ell/v](\ell) = v \overset{h}{\to} i = \gamma[y/i](y) = \gamma[y/i](g(\ell))$, therefore since g is an extension of h, $\mu[\ell/v](\ell) \overset{g}{\to} \gamma[y/i](g(\ell))$. This completes the proof of the lemma.

Using lemma 1, we get that $(g \circ [\sigma[x/\ell]])(a) = b[x/y]$ and $\mu[\ell/v] \xrightarrow{g} \gamma[y/i]$. But $g \circ (\sigma[x/\ell] :: \tau)(a\ \square) = (g \circ [\sigma[x/\ell]])(a)$, therefore $(a\ \square, \sigma[x/\ell] :: \tau, \mu[\ell/v]) \xrightarrow{g} \gamma[y/i]$, which completes this case.

Popping from the environment stack. If $s = v\ \square$ for some value v and $\Sigma = \sigma :: \tau$, then $d = (h \circ \Sigma)(v\ \square) = (h \circ \Sigma)(v)$, which is an irreducible term i such that $v \xrightarrow{h} i$, since:

- if v is a constant c, $i = (h \circ \Sigma)(c) = c$;
- if v is a closure $\{\lambda x.a, \sigma'\}$, $i = (h \circ \Sigma)(\{\lambda x.a, \sigma'\}) = (h \circ \sigma')(\lambda x.a)$ and $\mathsf{codom}\,\sigma' \subseteq \mathsf{dom}\,h$.

By definition of $\to_{\mathsf{cl}}$, $(t, \Pi, \nu) = (v, [\tau], \mu)$, and by reflexivity of $\to^*_{\mathsf{ca}}$, $\langle d, \gamma\rangle = \langle i, \gamma\rangle \to^*_{\mathsf{ca}} \langle i, \gamma\rangle$. But $i = (h \circ \Sigma(v))$ does not depend on Σ, therefore $i = (h \circ [\tau])(v)$. Moreover we know $\mu \xrightarrow{h} \gamma$ and with $g = h$, we get $(v, [\tau], \mu) \overset{g}{\sim} \langle i, \gamma\rangle$ which completes this case.

Variable Assignment. If $s = (x := v)$ for some variable x and value v and $\Sigma = [\sigma]$, then $d = (h \circ \Sigma)(x := v) = (y := i)$ with y a variable such that $y = (h \circ \sigma)(x)$ and $i = (h \circ \Sigma)(v)$. Therefore $(v, \sigma, \mu) \overset{h}{\sim} \langle i, \gamma\rangle$.

By definition of $\to_{\mathsf{cl}}$, $(s, \Pi, \nu) = ((), [\sigma], \mu[\sigma(x)/v])$, and by definition of $\to_{\mathsf{ca}}$, $\langle d, \gamma\rangle = \langle y := i, \gamma\rangle = \langle (), \gamma[y/i]\rangle$. The following lemma completes this case.

Lemma 2. $((), \sigma, \mu[\sigma(x)/v]) \overset{h}{\sim} \langle (), \gamma[y/i]\rangle$.

Proof. The domain conditions are fulfilled since $(v, \sigma, \mu) \overset{h}{\sim} \langle i, \gamma\rangle$, $\mathsf{dom}\,\mu = \mathsf{dom}\,\mu[\sigma(x)/v]$ and $\mathsf{dom}\,\gamma = \mathsf{dom}\,\gamma[y/i]$. Let $\ell \in \mathsf{dom}\,\mu[\sigma(x)/v] = \mathsf{dom}\,\mu$. If $\ell = \sigma(x)$ then $\mu[\sigma(x)/v](\ell) = v \overset{h}{\sim} i = \gamma[y/i](y) = \gamma[y/i](h(\ell))$ since $h(\ell) = (h \circ \sigma)(x) = (h \circ \sigma)(x) = y$. Otherwise $\mu[\sigma(x)/v](\ell) = \mu(\ell) \overset{h}{\sim} \gamma(h(\ell)) = \gamma[y/i](h(\ell))$ using that h is injective and h is an extension of h. Finally $() \xrightarrow{h} ()$, which completes the proof of the lemma.

Contexts. If $s = C[s_1]$ for some context C such that $(s_1, \Sigma, \mu) \to_{\mathsf{cl}} (t_1, \Pi, \nu)$, then by definition of $\to_{\mathsf{cl}}$, $t = C[t_1]$. By definition of $\overset{h}{\sim}$ there exists d_1 such that $d = C[d_1]$. By induction hypothesis there exists e_1, δ such that $\langle d_1, \gamma\rangle \to^*_{\mathsf{ca}} \langle e_1, \delta\rangle$ and $(t_1, \Pi, \nu) \overset{g}{\sim} \langle e_1, \delta\rangle$ for some g extension of h. By definition of $\to_{\mathsf{ca}}$, $\langle C[d_1], \gamma\rangle \to^*_{\mathsf{ca}} \langle C[e_1], \delta\rangle$. By induction on the structure of C, and using the fact that the context C cannot contain any $\square$, we can then prove that $(C[t_1], \Pi, \nu) \overset{g}{\sim} \langle C[e_1], \delta\rangle$, which completes this case.

Computing under the Pop Indicator $\square$. If $s = s_1\ \square$ for s_1 not a value, such that $(s_1, \Sigma, \mu) \to_{\mathsf{cl}} (t_1, \Pi, \nu)$, and $\Sigma = \Sigma' :: \sigma$, then by definition of $\to_{\mathsf{cl}}$, $t = t_1\ \square$ and $\Pi = \Pi' :: \sigma$ for some Π'. $(s_1\ \square, \Sigma' :: \sigma, \mu) \overset{h}{\sim} \langle d, \gamma\rangle$, therefore $(s_1, \Sigma', \mu) \overset{h}{\sim} \langle d, \gamma\rangle$. By induction hypothesis there exists e, δ such that $\langle d, \gamma\rangle \to^*_{\mathsf{ca}} \langle e, \delta\rangle$ and

$(t_1, \Pi, \nu) \overset{h}{\sim} \langle e, \delta \rangle$. Now this proves that $(t_1\ \square, \Pi' :: \sigma, \nu) \overset{h}{\sim} \langle e, \delta \rangle$, which completes this case.

□

Proof of Theorem 2. We proceed similarly as for the proof of Theorem 1, by induction on the derivation of $\langle d, \gamma \rangle \rightarrow_{\mathsf{ca}} \langle e, \delta \rangle$. We do not detail any case here. The cases for variable call, function application of a λ-term, variable assignment, and contexts are symmetric to the ones seen in the proof of Theorem 1. The case for function application of a constant function, composition, if conditional and while loop are straightforward inductive arguments. Finally, this Theorem does not need cases for λ-abstractions, popping from the environment stack or computing with the pop indicator, as no rule in $\rightarrow_{\mathsf{ca}}$ applies to those. □

5 Discussion

5.1 Capsules and Closures: A Strong Correspondence

Corollary 1 shows that capsules and closures are very strongly related. Not only there is a derivation based on capsules for every derivation based on closures, but these two derivations mirror each other. The computations are completely bisimilar, even though definining computations for capsules is simpler.

5.2 Capsules Allow to Suppress the Stack of Environments Σ

When using closures, a state is a triple (s, Σ, μ) whereas when using capsules, it is just a capsule $\langle e, \gamma \rangle$. If they are bisimilar under h, it means that $(h \circ \Sigma)(d) = e$ and $\mu \overset{h}{\rightarrow} \gamma$. Capsules eliminate the need for the stack of environments Σ and thus suppress the indirection in closures that was needed to handle imperative features. Their small-step semantics also does not need any stack of environments of any sort, making the state of computation much simpler. Finally, we originally created the capsule environment γ to replace the (closure) environments of Σ. However, it is remarkable that γ is much closer to the store μ, while at the same time eliminating the need for Σ.

Acknowledgements. We would like to thank Nikhil Swamy for suggesting this work during a presentation of [8].

References

1. Aboul-Hosn, K.: Programming with private state. Honors Thesis, The Pennsylvania State University (December 2001), `http://www.cs.cornell.edu/%7Ekamal/thesis.pdf`
2. Aboul-Hosn, K., Kozen, D.: Relational semantics of local variable scoping. Tech. Rep. 2005-2000, Cornell University (2005), `http://www.cs.cornell.edu/%7Ekamal/local.pdf`

3. Aboul-Hosn, K., Kozen, D.: Relational Semantics for Higher-Order Programs. In: Uustalu, T. (ed.) MPC 2006. LNCS, vol. 4014, pp. 29–48. Springer, Heidelberg (2006)
4. Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: LICS 1998: Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science, pp. 334–344. IEEE Computer Society, Washington, DC (1998)
5. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for idealized ALGOL with active expressions. Electr. Notes Theor. Comput. Sci. 3 (1996)
6. Felleisen, M., Hieb, R.: The revised report on the syntactic theories of sequential control and state. Theoretical Computer Science 103, 235–271 (1992)
7. Halpern, J.Y., Meyer, A.R., Trakhtenbrot, B.A.: The semantics of local storage, or what makes the free-list free? In: Proc. 11th ACM Symp. Principles of Programming Languages (POPL 1984), New York, NY, USA, pp. 245–257 (1984)
8. Jeannin, J.B.: Capsules and closures. In: Mislove, M., Ouaknine, J. (eds.) Proc. 27th Conf. Math. Found. Programming Semantics (MFPS XXVII). Elsevier Electronic Notes in Theoretical Computer Science, Pittsburgh, PA (2011)
9. Jeannin, J.B., Kozen, D.: Computing with capsules. Tech. Rep., Computing and Information Science, Cornell University (January 2011), `http://hdl.handle.net/1813/22082`
10. Laird, J.: A Game Semantics of Local Names and Good Variables. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 289–303. Springer, Heidelberg (2004)
11. Mason, I.A., Talcott, C.L.: References, local variables and operational reasoning. In: Seventh Annual Symposium on Logic in Computer Science, pp. 186–197. IEEE (1992), `http://www-formal.stanford.edu/MT/92lics.ps.Z`
12. Mason, I., Talcott, C.: Programming, transforming, and proving with function abstractions and memories
13. Mason, I., Talcott, C.: Axiomatizing operational equivalence in the presence of side effects. In: IEEE Fourth Annual Symposium on Logic in Computer Science, pp. 284–293. IEEE Computer Society Press (1989)
14. Mason, I., Talcott, C.: Equivalence in functional languages with effects (1991)
15. Milne, R., Strachey, C.: A Theory of Programming Language Semantics. Halsted Press, New York (1977)
16. Moggi, E.: Notions of computation and monads. Information and Computation 93(1) (1991)
17. Pitts, A.M.: Operationally-based theories of program equivalence. In: Dybjer, P., Pitts, A.M. (eds.) Semantics and Logics of Computation, pp. 241–298. Publications of the Newton Institute, Cambridge University Press (1997), `http://www.cs.tau.ac.il/~nachumd/formal/exam/pitts.pdf`
18. Pitts, A.M., Stark, I.D.B.: Operational reasoning in functions with local state. In: Gordon, A.D., Pitts, A.M. (eds.) Higher Order Operational Techniques in Semantics, pp. 227–273. Cambridge University Press (1998), `http://homepages.inf.ed.ac.uk/stark/operfl.pdf`
19. Pitts, A.M., Stark, I.D.B.: Observable Properties of Higher Order Functions that Dynamically Create Local Names, or What's New? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 122–141. Springer, Heidelberg (1993)
20. Scott, D.S.: Mathematical concepts in programmng language semantics. In: Proc. 1972 Spring Joint Computer Conferences, pp. 225–234. AFIPS Press, Montvale (1972)

21. Stoy, J.E.: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Cambridge (1981)
22. Sussman, G.J., Steele, G.L.: Scheme: A interpreter for extended lambda calculus. Higher-Order and Symbolic Computation 11, 405–439 (1998), http://dx.doi.org/10.1023/A:1010035624696, 10.1023/A:1010035624696
23. Winskel, G.: The Formal Semantics of Programming Languages. MIT Press (1993)

Nuprl as Logical Framework for Automating Proofs in Category Theory

Christoph Kreitz

Institut für Informatik, Universität Potsdam, 14482 Potsdam, Germany

Abstract. We describe the construction of a semi-automated proof system for elementary category theory using the Nuprl proof development system as logical framework. We have used Nuprl's display mechanism to implement the basic vocabulary and Nuprl's rule compiler to implemented a first-order proof calculus for reasoning about categories, functors and natural transformations. To automate proofs we have formalized both standard techniques from automated theorem proving and reasoning patterns that are specific to category theory and used Nuprl's tactic mechanism for the actual implementation. We illustrate our approach by automating proofs of natural isomorphisms between categories.

1 Introduction

Category theory [EM45] is a common framework for expressing abstract properties of mathematical structures that occur in many areas of mathematics and computer science. Abstract notions such as objects, morphisms, composition, identities, products, functors, transformations, duality, and isomorphisms are common to areas like set theory, logic, algebra, topology, semantics of programming languages, or formal software specification and development. The beauty of category theory is that it allows one to be completely precise about such concepts and that many algebraic constructions become exceedingly elegant at this level of abstraction. Diagrams can be used to illustrate essential insights and often make it unnecessary to provide further details of a proof, as these may be obtained entirely by standard patterns of reasoning.

However, since category theory is considerably more abstract than many other branches of mathematics, it becomes almost impossible to verify the details of such a proof. Readers frequently have to accept "obvious" assertions on faith, as complete proofs based on precise definitions often involve an enormous number of low-level details that must be checked. Furthermore, the high level of abstraction forces one to work in an atmosphere in which much of the intuition has been stripped away. As a result, the verification often becomes a matter of pure symbol manipulation, an area in which humans easily make mistakes.

On the other hand, proofs that rely on standard patterns of reasoning and symbol manipulation lend themselves well to automation. Providing such an automation serves several purposes. It enables users to generate completely formal proofs without having to go through all the details themselves, thus providing

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 124–148, 2012.

assurance that the statement is in fact true. It allows users to inspect details of a proof and get a better grasp of the standard patterns of reasoning in elementary category theory. It also shows that the proofs that many authors do not bother to provide actually may contain a tremendous amount of hidden detail and possibly even preconditions that the author might have taken for granted or overlooked entirely. Finally, it demonstrates that a proof is indeed trivial from an intellectual point of view, because it could be found automatically by a machine.

To provide a foundation for automating basic category theory reasoning Kozen [Koz04] presents a first-order axiomatization of elementary category theory and illustrates its use by giving a formal proof that the functor categories $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ are naturally isomorphic. Although the proof of this theorem omits many low-level details such as equality reasoning and simple first-order arguments, it is extremely long and required many hours of careful work to complete. To make sure that every detail of a proof can be validated it is necessary to implement the proof calculus and to develop proof strategies that support the automated construction of proofs by capturing the general patterns of reasoning used in hand-constructed proofs.

As platform for the implementation of Kozen's calculus we have selected the Nuprl system [CA+86]. Nuprl is a proof and programming environment for the interactive development of formalized mathematical knowledge as well as for the synthesis, verification, and optimization of software. Nuprl's current architecture [AC+00, Kre02, AB+05] is the product of many evolutions aimed at providing a theorem proving environment as rich and robust as its type theory. The resulting implementation composes a set of communicated processes, centered around a common knowledge base, called the *library*. The library contains definition objects, theorems, inference rules, and meta-level code (e.g. tactics), and serves as a transaction broker for the other processes. Those processes include user interfaces (*editors*), inference engines (*refiners*) and mechanisms for extracting programs from proofs, rewrite engines (*evaluators*), and *translators*. Translators between the formal knowledge stored in the library and, for instance, programming languages like Java or Ocaml [Kre04, KHH98] allow the formal reasoning tools to supplement real-world software from various domains and thus provide a *logical programming environment* for the respective languages.

While Nuprl was originally developed as theorem prover for Computational Type Theory [Con08], the current architecture has no predefined logic but uses formal library objects to define the syntax and inference rules of a logic. Thus the Nuprl system has become a *logical framework* that can accommodate arbitrary logics whose inference rules can be expressed in a sequent style. Although almost all of the actual development is still based on the Nuprl type theory, users may now embed entirely new theories as independent proof calculi into the system's library and use the framework to automate reasoning in these theories.

To make use of this potential of the Nuprl system, which had not been explored before, we proceeded as follows. To embed the vocabulary of elementary category theory we added abstract terms for each concept of the theory to the system's library as well as display forms for presenting these terms in a familiar

syntax. For all inference rules of Kozen's calculus we added rule objects to the library and used Nuprl's rule compiler to convert these into reasoning tactics that execute these rules in the proof environment. To automate reasoning we encoded standard theorem proving techniques as Nuprl proof tactics, developed additional tactics to capture the reasoning patterns that are specific to category theory, and added these tactics as code objects to the library.

In [KKR06] we have demonstrated that this approach can in fact automate Kozen's proof of the isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D}, \mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C}, \mathsf{Fun}[\mathsf{D}, \mathsf{E}]]$. With the additional techniques and calculi described in this paper, we are now able to construct fully automated proofs for a variety of isomorphisms between categories as well as proofs of the naturality of all all these isomorphisms.

The structure of this paper follows the development outlined above. In Section 2 we briefly review Kozen's first-order axiomatization of elementary category theory. We then describe the implementation of this calculus within the Nuprl logical framework in Section 3. Strategies that encode standard techniques from automated theorem proving will be presented in Section 4. Strategies that automate reasoning specific to category theory and their application to proofs of natural isomorphisms will be discussed in Section 5. We conclude by discussing related approaches, insights that we have observed in the course of this work, and new research issues that result from these observations.

2 An Axiomatization of Elementary Category Theory

We assume the reader to be familiar with the basic definitions and notation of category theory [BW90, McL71]. We begin our review of Kozen's calculus [Koz04] with a few notational conventions.

- Symbols in sans serif, such as C, always denote categories. The category Cat is the category of (small) categories and functors.
- If C is a category, the symbol C denotes both the category C and the set of *objects* of C.
- $A : \mathsf{C}$ indicates that A is an object of the category C. Composition is denoted by the symbol $\circ$ and the identity on object $A : \mathsf{C}$ is denoted 1_A.
- $h : \mathsf{C}(A, B)$ indicates that h is an *arrow* of the category C with *domain* A and *codomain* B.
- $\mathsf{Fun}[\mathsf{C}, \mathsf{D}]$ denotes the functor category whose objects are the *functors* from C to D and whose arrows are the *natural transformations* on such functors. Thus $F : \mathsf{Fun}[\mathsf{C}, \mathsf{D}]$ indicates that F is a functor from C to D and $\varphi : \mathsf{Fun}[\mathsf{C}, \mathsf{D}](F, G)$ indicates that φ is a natural transformation with domain F and codomain G, where $F, G : \mathsf{Fun}[\mathsf{C}, \mathsf{D}]$.
- F^1 and F^2 denote the object and arrow components, respectively, of a functor F. Thus if $F : \mathsf{Fun}[\mathsf{C}, \mathsf{D}]$, $A, B : \mathsf{C}$, and $h : \mathsf{C}(A, B)$, then $F^1A, F^1B : \mathsf{D}$ and $F^2h : \mathsf{D}(F^1A, F^1B)$.
- Function application binds tighter than the operators ^{1} and ^{2}. Thus the expression F^1A^2 should be parsed $(F^1A)^2$.

- C^{op} denotes the opposite category of C.
- $\mathsf{C}\times\mathsf{D}$ denotes the product of the categories C and D. Its objects are pairs $(A, X) : \mathsf{C}\times\mathsf{D}$, where $A : \mathsf{C}$ and $X : \mathsf{D}$, and its arrows consist of pairs $(f, h) : (\mathsf{C} \times \mathsf{D})((A, X), (B, Y))$, where $f : \mathsf{C}(A, B)$ and $h : \mathsf{D}(X, Y)$. Composition and identities are defined component-wise; that is,

$$(g, k)\circ(f, h) \stackrel{\text{def}}{=} (g\circ f, k\circ h) \tag{1}$$

$$1_{(A,X)} \stackrel{\text{def}}{=} (1_A, 1_X). \tag{2}$$

Inference rules are based on sequents $\Gamma \vdash \alpha$, where Γ is a type environment (a set of type judgments on atomic symbols) and α is either a type judgment or an equation. The rules cover the basic properties of categories, functors and natural transformations. They are divided into symmetric sets of rules for analysis (elimination) and synthesis (introduction). There are also rules for equational reasoning. To support first-order reasoning about higher-order concepts like functors, the rules deal with their first-order components.

Categories. There is a collection of rules covering the basic properties of categories, which are essentially the rules of typed monoids. These rules include typing rules for composition and identities as well as equational rules for associativity and two-sided identity.

$$\frac{\Gamma \vdash A, B, C : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A, B), \quad \Gamma \vdash g : \mathsf{C}(B, C)}{\Gamma \vdash g\circ f : \mathsf{C}(A, C)} \tag{3}$$

$$\frac{\Gamma \vdash A : \mathsf{C}}{\Gamma \vdash 1_A : \mathsf{C}(A, A)} \tag{4}$$

$$\frac{\Gamma \vdash A, B, C, D : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A, B), \quad \Gamma \vdash g : \mathsf{C}(B, C), \quad \Gamma \vdash h : \mathsf{C}(C, D)}{\Gamma \vdash (h\circ g)\circ f = h\circ(g\circ f)}, \tag{5}$$

$$\frac{\Gamma \vdash A, B : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A, B)}{\Gamma \vdash f\circ 1_A = f} \qquad \frac{\Gamma \vdash A, B : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A, B)}{\Gamma \vdash 1_B\circ f = f} \tag{6}$$

Functors. A functor F from C to D is determined by its object and arrow components F^1 and F^2. The components must be of the correct type and must preserve composition and identities. These properties are captured in the following rules.

Analysis

$$\frac{\Gamma \vdash F : \mathsf{Fun}[\mathsf{C}, \mathsf{D}], \quad \Gamma \vdash A : \mathsf{C}}{\Gamma \vdash F^1 A : \mathsf{D}} \tag{7}$$

$$\frac{\Gamma \vdash F : \mathsf{Fun}[\mathsf{C}, \mathsf{D}], \quad \Gamma \vdash A, B : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A, B)}{\Gamma \vdash F^2 f : \mathsf{D}(F^1 A, F^1 B)} \tag{8}$$

$$\frac{\Gamma \vdash F : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma \vdash A, B, C : \mathsf{C}, \quad \Gamma \vdash f : \mathsf{C}(A,B), \quad \Gamma \vdash g : \mathsf{C}(B,C)}{\Gamma \vdash F^2(g \circ f) = F^2 g \circ F^2 f} \tag{9}$$

$$\frac{\Gamma \vdash F : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma \vdash A : \mathsf{C}}{\Gamma \vdash F^2 1_A = 1_{F^1 A}} \tag{10}$$

Synthesis

$$\frac{\begin{array}{r}\Gamma,\ A : \mathsf{C} \ \vdash\ F^1 A : \mathsf{D} \\ \Gamma,\ A, B : \mathsf{C},\ g : \mathsf{C}(A,B) \ \vdash\ F^2 g : \mathsf{D}(F^1 A, F^1 B) \\ \Gamma,\ A, B, C : \mathsf{C},\ f : \mathsf{C}(A,B),\ g : \mathsf{C}(B,C) \ \vdash\ F^2(g \circ f) = F^2 g \circ F^2 f \\ \Gamma,\ A : \mathsf{C} \ \vdash\ F^2 1_A = 1_{F^1 A}\end{array}}{\Gamma \ \vdash\ F : \mathsf{Fun}[\mathsf{C},\mathsf{D}]} \tag{11}$$

Natural Transformations. A natural transformation $\varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G)$ is a function that for each object $A : \mathsf{C}$ gives an arrow $\varphi A : \mathsf{D}(F^1 A, G^1 A)$, called the *component* of φ at A, such that for all arrows $g : \mathsf{C}(A,B)$, the following diagram commutes:

$$\begin{array}{ccc} F^1 A & \xrightarrow{F^2 g} & F^1 B \\ {\scriptstyle \varphi A}\downarrow & & \downarrow{\scriptstyle \varphi B} \\ G^1 A & \xrightarrow{G^2 g} & G^1 B \end{array} \tag{12}$$

Composition and identities are defined by

$$(\varphi \circ \psi) A \stackrel{\text{def}}{=} \varphi A \circ \psi A \tag{13}$$

$$1_F A \stackrel{\text{def}}{=} 1_{F^1 A}. \tag{14}$$

The property (12), along with the typing of φ, are captured in the following rules.

Analysis

$$\frac{\Gamma \vdash \varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G)}{\Gamma \vdash F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}]} \tag{15}$$

$$\frac{\Gamma \vdash \varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G), \quad \Gamma \vdash A : \mathsf{C}}{\Gamma \vdash \varphi A : \mathsf{D}(F^1 A, G^1 A)} \tag{16}$$

$$\frac{\Gamma \vdash \varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G), \quad \Gamma \vdash A, B : \mathsf{C}, \quad \Gamma \vdash g : \mathsf{C}(A,B)}{\Gamma \vdash \varphi B \circ F^2 g = G^2 g \circ \varphi A} \tag{17}$$

Synthesis

$$\frac{\begin{array}{r}\Gamma \ \vdash\ F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}] \\ \Gamma,\ A : \mathsf{C} \ \vdash\ \varphi A : \mathsf{D}(F^1 A, G^1 A) \\ \Gamma,\ A, B : \mathsf{C},\ g : \mathsf{C}(A,B) \ \vdash\ \varphi B \circ F^2 g = G^2 g \circ \varphi A\end{array}}{\Gamma \ \vdash\ \varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G)} \tag{18}$$

Equational Reasoning. Besides the usual domain-independent axioms of typed equational logic (reflexivity, symmetry, transitivity, and congruence), certain domain-dependent equations on objects and arrows are assumed as axioms, including the associativity of composition (5) and two-sided identity rules (6) for arrows, the equations (1) and (2) for products, and the equations (13) and (14) for natural transformations. There are also extensionality rules for objects of functional type:

$$\frac{\Gamma \vdash F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma, A : \mathsf{C} \vdash F^1 A = G^1 A}{\Gamma \vdash F^1 = G^1} \tag{19}$$

$$\frac{\Gamma \vdash F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma,\ A, B : \mathsf{C},\ g : \mathsf{C}(A,B) \vdash F^2 g = G^2 g}{\Gamma \vdash F^2 = G^2} \tag{20}$$

$$\frac{\Gamma \vdash F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma \vdash F^1 = G^1, \quad \Gamma \vdash F^2 = G^2}{\Gamma \vdash F = G} \tag{21}$$

$$\frac{\Gamma \vdash F, G : \mathsf{Fun}[\mathsf{C},\mathsf{D}], \quad \Gamma \vdash \varphi, \psi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G), \quad \Gamma,\ A : \mathsf{C} \vdash \varphi A = \psi A}{\Gamma \vdash \varphi = \psi} \tag{22}$$

Equations on types and substitution of equals for equals in type expressions are also permitted. Any such equation $\alpha = \beta$ takes the form of a rule

$$\frac{\Gamma \vdash A : \alpha}{\Gamma \vdash A : \beta}. \tag{23}$$

For the application example, the following type equations are postulated

$$\mathsf{Cat}(\mathsf{C},\mathsf{D}) = \mathsf{Fun}[\mathsf{C},\mathsf{D}] \tag{24}$$

$$\mathsf{C}^{\mathsf{op}} = \mathsf{C} \tag{25}$$

$$\mathsf{C}^{\mathsf{op}}(A,B) = \mathsf{C}(B,A). \tag{26}$$

Other Rules. There are also various rules for products, weakening, and other structural rules for manipulation of sequents. These are all quite standard and do not bear explicit mention.

3 Implementing the Proof Calculus in Nuprl

Kozen's axiomatization is sufficient for the development of completely formal proofs for all theorems in elementary category theory. Kozen [Koz04] illustrates this fact by providing a rigorous formal proof that the functor categories $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ are naturally isomorphic. Although the proof is fairly straightforward and omits many details for the sake of readability it takes 13 pages on paper. As proofs of that size are difficult to construct and carry the potential for errors it is necessary to implement the proof system and to automate reasoning steps that mathematicians would consider obvious. In this section we will show how the Nuprl logical framework can be used to rapidly construct an implementation of Kozen's calculus.

3.1 Embedding the Vocabulary

We made use of Nuprl's definition mechanism to implement the vocabulary of elementary category theory. *Abstraction objects* can be used to add new abstract terms to the formal library whose meaning may either be defined through expressions of the formal language defined so far, or be left unspecified if only the signature shall be fixed. The abstract term for the set of objects of a category C, for instance, is implemented by adding an abstraction object named Obj to the library, which in the library listing appears as follows.

```
A  Obj          Obj{}(.C) == !primitive
```

This object introduces a new abstract term Obj with one subterm, denoted by the variable C, and defines it to be primitive.[1]

Display forms can then be used to make the visual appearance of abstract terms conform to the notation used on paper without changing their internal structure. According to the conventions in section 2, for instance, the set of objects of a category C is denoted by C. To introduce this notation, we add a display object named Obj_df to the library.

```
D  Obj_df       <C>   ≡ Obj{}(.<C>)
```

This object makes sure that the abstract term Obj{}(.C) will be displayed as C. The angle brackets around C indicate that C is a parameter of the display form.

Display forms are important for interaction between the system and human users as well as for a readable presentation of the implemented theory on paper. The reasoning system itself deals only with abstract terms and can therefore easily distinguish between a category and the set of its objects although both have the same representation on the screen.

We have created abstraction and display objects for each concept of elementary category theory. Although it is possible to represent these concepts in terms of Nuprl's Type Theory and validate the implemented inference rules on this basis we have chosen to use Nuprl only as logical framework for building an independent system for reasoning about category theory and have declared all fundamental abstract terms to be primitive.

Figure 1 lists the display objects that implement the vocabulary of elementary category theory. For the sake of readability we use math-font instead of angle brackets to denote parameters. Note that a composition $g \circ f$ of two arrows depends on the category C to which f and g belong but C is never mentioned explicitly in compositions. Accordingly, C has to occur in the abstract term but should not be shown when a composition is being displayed. For this reason, the parameter C does not occur on the left hand side of the display form for compositions and identities and the display of the category will be suppressed as soon as the complete term has been entered into the system.

[1] The formal declaration of Obj also contains an empty list of parameters between curly braces and an empty list of variable bindings for the subterm in front of the dot before C. Parameters and variable bindings are features of the Nuprl logical framework that are necessary for representing more expressive theories but are not needed for implementing elementary category theory.

Obj	C	$\equiv$	Obj{}(.C)
Mor	$C(A,B)$	$\equiv$	Mor{}(.C;.A;.B)
Comp	$(g \circ f)$	$\equiv$	Comp{}(.C;.g;.f)
Id	$1A$	$\equiv$	Id{}(.C;.A)
fun1	F^1A	$\equiv$	fun1{}(.F;.A)
fun2	F^2g	$\equiv$	fun2{}(.F;.g)
CatFun	Fun[C,D]	$\equiv$	CatFun{}(.C;.D)
CatProd	$C \times D$	$\equiv$	CatProd{}(.C;.D)
CatOp	C-op	$\equiv$	CatOp{}(.C)
CatCat	Cat	$\equiv$	CatCat{}()

Fig. 1. Display objects implementing the syntax of elementary category theory

Currently, Nuprl's display is restricted to a single 8-bit font. This limits the use of symbols, subscripts and superscripts to fixed characters. Identities, usually written as 1_A or $1_{(A,X)}$, have to be presented as 1A and 1<A,X>.[2] Apart from these restrictions, all the basic category-theoretic vocabulary will be displayed in the same way as described in Section 2.

Besides the vocabulary of elementary category theory Kozen [Koz04] uses a notion of isomorphism of categories and naturality of isomorphisms. These concepts can be defined in terms of the existing notions (see Section 5.2 for an explanation) and are implemented by the following formal definitions.

F and G are inverse
  == $\forall$A,B:C. $\forall$f:C(A,B). G^1F^1A=A $\in$ C $\wedge$ G^2F^2f=f $\in$ C(A,B)
    $\wedge$ $\forall$X,Y:D. $\forall$h:D(X,Y). F^1G^1X=X $\in$ D $\wedge$ F^2G^2h=h $\in$ D(X,Y)

$C \hat{=} D$
  == $\exists\theta$:Fun[C,D]. $\exists\eta$:Fun[D,C]. θ and η are inverse

$C \hat{=} D$ via θ and η
  == $\theta \in$ Fun[C,D] $\wedge$ $\eta \in$ Fun[D,C] $\wedge$ θ and η are inverse

C and D are naturally isomorphic
  == $\exists$CAT. $\exists$U,V:Fun[CAT,Cat]
    $\exists\theta$:Fun[CAT,Cat](U,V). $\exists\eta$:Fun[CAT,Cat](V,U).
      $\forall$c:CAT. $C \hat{=} D$ via θc and ηc

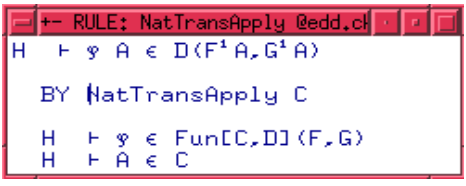
3.2 Implementation of Inference Rules

Like the proof calculus presented in the previous section, Nuprl's inference mechanism is based on sequents. Nuprl's reasoning style, however, is goal-oriented, which means that inference rules operate top-down, refining a goal sequent into a set of subgoal sequents. Inference rules therefore have to be rephrased in a top-down fashion before they can be added to the system.

For the actual implementation of the proof calculus we made use of Nuprl's rule mechanism. *Rule objects* can be used to add schematic inference rules to the formal library. These consist of formal terms that describe a goal sequent and

[2] In Nuprl pairs use angle brackets <A,X> instead of parentheses.

the corresponding subgoal sequents and may contain pattern variables that can be matched against the components of the actual goal sequent in a proof. Since the representation of the rules in the system is identical to the paper version it is easy to check the faithfulness of the implementation. Rule (16), for instance, is represented by a rule object `NatTransApply` with the following contents.



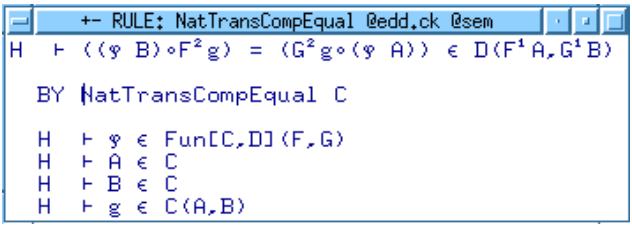
The rule states that in order to prove a goal sequent $\Gamma \vdash \varphi A : \mathsf{D}(F^1 A, G^1 A)$ one has to prove $\Gamma \vdash \varphi : \mathsf{Fun}[\mathsf{C},\mathsf{D}](F,G)$ and $\Gamma \vdash A : \mathsf{C}$ for some category C, which is exactly the same as rule (16). Due to the top-down style of the rule, C must be provided as parameter, since it occurs in the subgoal sequents but not in the main goal.

To create the actual inference rule from its representation as term one applies the *rule compiler* of the Nuprl logical framework to the rule object. This generates a proof tactic that matches the first line of a rule object against the actual goal sequent of a proof and creates the subgoal sequents by instantiating the lines below the name of the rule accordingly. The proof tactic for the above rule, for instance, is generated by the following simple ML declaration

```
let NatTransApply C = Refine 'NatTransApply' [term_arg C].
```

To apply this tactic, one has to provide a term C as argument, which is then inserted into the two subgoals created by the rule. Tactics may also expect tokens as arguments, which will then be used as names for variables that occur in the subgoals but not in the main goal. The tactic for the synthesis rule (11), for instance, requires five such names (for A, B, C, f, and g) to be provided.

Since Nuprl supports typed equalities and types often provide useful information for guiding proofs, we added types to all the inference rules that deal with equalities. For example, rule (17) is represented as follows:



We have generated rule objects for all the rules described in Section 2, as well as rules for dealing with products. Logical rules and rules dealing with extensional equality and substitution are already provided by Nuprl.

4 Automating First-Order and Equational Reasoning

The implementation of the proof calculus described in Section 3 enables us to create formal proofs for many theorems of basic category theory. But even the most simple of these theorems already lead to proofs with hundreds or even thousands of inference steps, as illustrated in [Koz04]. Since most of these statements are considered mathematically trivial, it should be possible to completely automate their proofs.

We have developed a small collection of strategies for automated proof search in basic category theory. Some of these strategies are based on generic techniques from automated theorem proving. Others are intended to capture the general patterns of category theory specific reasoning that we have observed in hand-constructed proofs. We will discuss the former in this section and elaborate on the latter in Section 5.

Most of the inference rules of our proof calculus are simple refinement rules that describe how to decompose a proof obligation into simpler components. Given a specific proof goal, there are only few rules that can be applied at all. Thus to a large extent, proof search consists of determining applicable rules and their parameters from the context, applying the rule, and then continuing the search on all the subgoals. Occasionally we will have to prove equalities, which may involve the application of extensionality rules as well as standard equality reasoning.

4.1 Automating Search

To support proof automation, all basic inference rules first had to be converted into simple tactics that automatically determine the parameters of these rules.

Generating names for new variables in the subgoals, as in the case of the extensionality rules (19)–(22), is straightforward. In principle it is sufficient to use a procedure that generate arbitrary new names but for the sake of readability we had the procedure generate memnonic names that fit the textual description of the rules.

To determine the terms that have to provided as parameters for certain inference rules one can take advantage of the fact that these parameters are explicitly mentioned in the subgoals of the rule, which puts certain type constraints on possible values. In the rule `NatTransApply`, for instance, C is the category to which the object `A` belongs and also the domain of the functors `F` and `G`. Therefore all term parameters of inference rules can be determined through an *extended type inference* algorithm.

To identify applicable rules it is sufficient to analyze the terms and types in the conclusion of the goal sequent. A conclusion of the form $\varphi A \in \mathsf{D}(X, Y)$, for instance, suggests the application of the rule `NatTransApply` or, less likely, of the rule `NatTransFormation` (rule (18)) if D turns out to be a functor category. In most cases only one rule can be meaningfully applied to a proof goal with a type judgment and this rule can be identified with the help of extended type inference.

An important issue is *loop control*. Since the synthesis rules for functors and natural transformations are the inverse of the corresponding analysis rules, an analysis rule could create a subgoal that has already been decomposed by a synthesis rule before and thus create a looping argument. To prevent such loops we made each proof branch keep track of proof goals to which a synthesis rule had been applied. Analysis rules that would generate one of these goals as a subgoal will thus be blocked from being applied.

4.2 Equality Reasoning

Equality reasoning is a key component in formal category-theoretic proofs. Ten of the inference rules deal with equalities and can be used to replace a term by one that is semantically equal. Since equality rules can be used both ways, they are a very powerful tool in the hands of a skilled user, but a potential cause for loops in an automated search for a proof. A simple proof search method as described above is therefore insufficient for automating proofs involving equalities.

We have decided to base our proof search mechanism on rewriting. For the purpose of *finding* a proof for a given equality we assign a direction to each of the equalities and attempt to rewrite terms into some normal form. Furthermore, the search procedure has to keep track of the types involved in these equalities, which are sometimes crucial for finding a proper match and, as in the case of rule (17), for determining the right-hand side of an equality from the left-hand side. The inference rules described in Section 2, including those dealing with associativity and identity, lead to the following typed rewrites.

Rewrite			**Type**	**Rule**
$\langle g, k\rangle \circ \langle f, h\rangle$	$\mapsto$	$\langle g\circ f, k\circ h\rangle$	$\mathsf{C}\times\mathsf{D}(\langle A_1, X_1\rangle, \langle A_3, X_3\rangle)$	(01)
$1_{\langle A,X\rangle}$	$\mapsto$	$\langle 1_A, 1_X\rangle$	$\mathsf{C}\times\mathsf{D}(\langle A, X\rangle, \langle A, X\rangle)$	(02)
$1_B \circ f$	$\mapsto$	f	$\mathsf{C}(A, B)$	(06a)
$f\circ 1_A$	$\mapsto$	f	$\mathsf{C}(A, B)$	(06b)
$h\circ(g\circ f)$	$\mapsto$	$(h\circ g)\circ f$	$\mathsf{C}(A, B_2)$	(05)
$F^2(g\circ f)$	$\mapsto$	$F^2 g\circ F^2 f$	$\mathsf{D}(F^1 A, F^1 B_1)$	(09)
$F^2 1_A$	$\mapsto$	$1_{F^1 A}$	$\mathsf{D}(F^1 A, F^1 A)$	(10)
$(\psi\circ\varphi)A$	$\mapsto$	$\psi A\circ\varphi A$	$\mathsf{D}(F^1 A, H^1 A)$	(13)
$1_F A$	$\mapsto$	$1_{F^1 A}$	$\mathsf{D}(F^1 A, F^1 A)$	(14)
$\varphi B\circ F^2 g$	$\mapsto$	$G^2 g\circ\varphi A$	$\mathsf{D}(F^1 A, G^1 B)$	(17)

Each rewrite is executed by applying a substitution, which is validated by applying the corresponding equality rule mentioned in the table above. The equations (24)–(26) deal solely with types and are treated separately.

The above rewrite system is incomplete, as it cannot prove the equality of terms like $F^2(1_A, 1_X)$ and $1_{F^1(A,X)}$ that can be shown equal with the inference rules. To convert the equational theory contained in our calculus into an equivalent set of rewrite rules guaranteeing normalization and confluence, we have applied the superposition-based Knuth-Bendix completion procedure [EB70]. As a result, the following typed rewrites were added to the system.

Rewrite			Type	Rules
$F^2\langle 1_A, 1_X\rangle$	$\mapsto$	$1_{F^1\langle A,X\rangle}$	$\mathsf{E}(F^1\langle A,X\rangle, F^1\langle A,X\rangle)$	(02),(10)
$F^2\langle g,k\rangle \circ F^2\langle f,h\rangle$	$\mapsto$	$F^2\langle g\circ f, k\circ h\rangle$	$\mathsf{E}(F^1\langle A,X\rangle, F^1\langle C,X\rangle)$	(01),(09)
$(\varphi Y A)\circ(F^2 g A)$	$\mapsto$	$(G^2 g A)\circ(\varphi X A)$	$\mathsf{E}(F^1X^1A, G^1Y^1A)$	(17),(13)
$(\varphi Y\circ\psi Y)\circ F^2 g$	$\mapsto$	$(G^2 g\circ\varphi X)\circ\psi X$	$\mathsf{E}(F^1X, G^1Y)$	(13),(17)
$H^2(\varphi Y)\circ H^2(F^2 g)$	$\mapsto$	$H^2(G^2 g)\circ H^2(\varphi X)$	$\mathsf{E}(H^1F^1X, H^1G^1Y)$	(05),(17)
$(h\circ\varphi Y)\circ F^2 g$	$\mapsto$	$(h\circ G^2 g)\circ\varphi X$	$\mathsf{D}(F^1X, Z)$	(09),(17)
$((h\circ G^2 g)\circ\varphi X)\circ\psi X$	$\mapsto$	$((h\circ\varphi Y)\circ\psi Y)\circ F^2 g$	$\mathsf{E}(F^1X, Z)$	(05),(13),(17)
$(h\circ H^2(\varphi Y))\circ H^2(F^2 g)$	$\mapsto$	$(h\circ H^2(G^2 g))\circ H^2(\varphi X)$	$\mathsf{E}(H^1F^1X, Z)$	(09),(09),(17)

Once a set of rewrites for a given equality has been found it is converted into a series of refinement steps by applying the equality rules associated with each rewrite in the appropriate direction. As a result, the generated proof tree contains a trace of the chain of equalities used, which can then be inspected by a human user interested in understanding the details of a proof.

4.3 Performance Issues

One of the disadvantages of refinement style reasoning is that proof trees may contain identical proof goals in different branches. This is especially true after the application of synthesis and extensionality rules, which must be used quite often in complex proofs.

```
H ⊢ F ∈ Fun[C,D]
 BY FunFormation A B B1 f g
   H, A:C ⊢ F¹A ∈ D
   H, A:C, B:C, f:C(A,B) ⊢ F²f ∈ D(F¹A,F¹B)
   H, A:C, B:C, B1:C, f:C(A,B), g:C(B,B1) ⊢ F²(g∘f) = F²g∘F²f ∈ D(F¹A,F¹B1)
   H, A:C ⊢ F²1A = 1F¹A ∈ D(F¹A,F¹A)
```

The rule `FunFormation` (rule (11)), for instance, generates a subgoal of the form F^1A, which will eventually reappear in the proof of the second, since F^1A occurs within the type of that goal. Furthermore, the first two subgoals will also reappear in the proofs of the third and fourth subgoals. In a bottom-up proof, one would prove these goals only once and reuse them whenever they are needed to complete the proof of another goal while a standard refinement proof would require us to prove the same goal over and over again.

Obviously we could optimize the corresponding rules for top-down reasoning and simply drop the redundant subgoals. But this would mean deviating from the original proof calculus. If one intends to retain faithfulness these rules must remain unchanged. Instead, we have wrapped the corresponding tactic with a controlled application of the cut rule: we simply assert a generalization of the first two subgoals of rule (11) before applying the rule. As a result they appear in the hypothesis list of the all subgoals and have to be proved only once.

Although this method is a fairly simple trick, it leads to an astonishing reduction in the size of automatically generated proofs. A complete proof of the isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D}, \mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C}, \mathsf{Fun}[\mathsf{D}, \mathsf{E}]]$ (see Section 5) without cuts consists of almost 30,000 inference steps. After introducing the wrapper the size of the proof was reduced to only 3,000 steps.

4.4 First-Order Reasoning about Higher Order Objects

Although elementary category theory deals with higher-order objects such as functors and natural transformations, Kozen's axiomatization [Koz04] has been formulated entirely as first-order calculus. This means that the properties of functors and natural transformations have to be described in terms of their first-order components (rules (7) – (11), (15) – (18)).

Keeping the reasoning level first order becomes more difficult when reasoning about isomorphisms between categories. Two categories C and D are isomorphic, denoted by $\mathtt{C} \mathrel{\hat{=}} \mathtt{D}$, if there are two functors $\theta : \mathsf{Fun}[\mathsf{C}, \mathsf{D}]$ and $\eta : \mathsf{Fun}[\mathsf{D}, \mathsf{C}]$ that are inverses of each other. A computerized proof of this fact would require us to provide θ and η, which involves higher-order reasoning.

To avoid this issue, the proof in [Koz04] specifies the object and arrow components $\theta^1 A$ and $\theta^2 f$ for A an object of C and f an arrow of C through first-order equations. If these components are again functors or natural transformations, one has to specify subcomponents until the first-order level has been reached. In the proof of the isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D}, \mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C}, \mathsf{Fun}[\mathsf{D}, \mathsf{E}]]$, the following four equations are needed to specify θ:

$$\begin{aligned}
\theta^1 \mathtt{F}^1 \mathtt{A}^1 \mathtt{X} &\equiv \mathtt{F}^1\mathtt{<A,X>} \\
\theta^1 \mathtt{F}^1 \mathtt{A}^2 \mathtt{k} &\equiv \mathtt{F}^2\mathtt{<1A,k>} \\
\theta^1 \mathtt{F}^2 \mathtt{f\ X} &\equiv \mathtt{F}^2\mathtt{<f,1X>} \\
\theta^2\, \varphi\ \mathtt{X\ X1} &\equiv \varphi\ \mathtt{<X,X1>}
\end{aligned}$$

Mathematically, these four equations are sufficient for the proof, since any functor satisfying these equations can be used to complete the proof. In a computerized formal proof, however, we also have to prove the existence of a functor satisfying these equations. Constructing the functor from the equations is straightforward if it is uniquely specified by them. It is only necessary to assemble the respective object and arrow (sub-)components into a single closed functor object. Since assembling the functor from components has nothing to do with the main proof, this step is performed automatically in the background as soon as the components have been completely specified.

5 Automating Reasoning Specific to Category Theory

The mechanisms described in the previous section are sufficient to verify properties of given functors and natural transformations. A proof of the isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D}, \mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C}, \mathsf{Fun}[\mathsf{D}, \mathsf{E}]]$ can be completely automated once the specifications of the inverse functors θ and η have been provided. One only has to unfold the definition of functors being inverse to each other and then all the remaining steps are straightforward for the automated proof search procedure `AutoCAT2` and take only a few seconds to complete.

```
* ∀C,D,E:Categories.  Fun[C×D,E] ≙ Fun[C,Fun[D,E]]
  BY ....
  1.-3. C,D,E: Cat
  4.  θ¹F¹A¹X ≡ F¹<A,X>  ∧ θ¹F¹A²k  ≡ F²<1A,k>
   ∧ θ¹F²f X ≡ F²<f,1X> ∧ θ² φ X X1 ≡ φ <X,X1>
  5.  η¹F¹<A,X>  ≡ F¹A¹X ∧ η¹F²<f,g> ≡ ((F²f cod(g))∘F¹dom(f)²g)
   ∧ η²φ <A,X1> ≡ φ A X1
  ⊢ θ and η are inverse
  BY AutoCAT2
```

But finding the specifications of the functors cannot be accomplished with standard reasoning techniques, since matching and unification are of little help here. On the other hand, for a trained mathematician this is a trivial task as there are only a few "obvious" choices. If a functor exists at all the types of its first-order components usually contain all the information that is needed to make an educated guess. In practice, this simple heuristic hardly ever fails, particularly if the proof is considered trivial from an intellectual point of view.

Since the proof steps that are considered intellectually trivial should be automated, we have developed heuristics that attempt to determine the most obvious specifications for functors or natural transformations of a given type. We will illustrate both by example of the isomorphism between the categories Fun[C×D, E] and Fun[C, Fun[D, E]] and the naturality of this isomorphism and then discuss a few details of their formalization as implemented proof strategy.

5.1 Finding Witnesses for Isomorphisms between Categories

To prove the existence of a functor F between two categories C and D our heuristic first generates typing subgoals for all first-order components of the functor. For this purpose it applies the refinement rules of our proof calculus to the goal $\Gamma \vdash$ F ∈ Fun[C,D], where Γ is the current context of the proof and F is a new variable, and proceeds with refining typing judgments until they cannot be decomposed anymore. Equalities will be ignored as they do not provide information that is immediately useful.

To prove the existence of a functor θ between the categories Fun[C×D, E] and Fun[C, Fun[D, E]], for instance, the application of refinement rules yields the (incomplete) proof shown in figure 2. The four open subgoals in this proof, labelled 1.1.1, 1.1.2, 1.2.1, and 2.1.1, describe the typing conditions for all the first-order components of θ.

Next, the heuristic tries to determine a term that satisfies the given type judgment in the corresponding type environment. Because this term is intended to be a "trivial" solution, it should use be built solely from parameters explicitly mentioned in the first-order component of the functor and constructs that mathematicians would consider obvious choices like functor application, identities, domains, ranges, pairs etc. Obviously, the heuristic has to rely on type inference to construct a term that fits these requirements.

To solve subgoal 1.1.1., for instance, the heuristic has to construct an object of the category E from the components F : Fun[C×D, E], A : C, and X : D.

```
top     ⊢ θ ∈ Fun[Fun[C×D,E],Fun[C,Fun[D,E]]]                          (11)
1.      F:Fun[C×D,E]        ⊢ θ¹F ∈ Fun[C,Fun[D,E]]                    (11)
1.1.    F:Fun[C×D,E], A:C ⊢ θ¹F¹A ∈ Fun[D,E]                           (11)
1.1.1.  F:Fun[C×D,E], A:C, X:D
                            ⊢ θ¹F¹A¹X ∈ E
1.1.2.  F:Fun[C×D,E], A:C, X,Y:D, k:D(X,Y)
                            ⊢ θ¹F¹A²k ∈ E(θ¹F¹A¹X, θ¹F¹A¹Y)
1.2.    F:Fun[C×D,E], A,B:C, f:C(A,B)
                            ⊢ θ¹F²f ∈ Fun[D,E](θ¹F¹A,θ¹F¹B)            (18)
1.2.1.  F:Fun[C×D,E], A,B:C, f:C(A,B), X:D
                            ⊢ θ¹F²f X ∈ E(θ¹F¹A¹X, θ¹F¹B¹X)
2.      F,G:Fun[C×D,E], φ:Fun[C×D,E](F,G)
                            ⊢ θ² φ ∈ Fun[C,Fun[D,E]](θ¹F, θ¹G)          (18)
2.1.    F,G:Fun[C×D,E], φ:Fun[C×D,E](F,G), X:C
                            ⊢ θ² φ X ∈ Fun[D,E](θ¹F¹X, θ¹G¹X)           (18)
2.1.1.  F,G:Fun[C×D,E], φ:Fun[C×D,E](F,G), X:C, X1:D
                            ⊢ θ² φ X X1 ∈ E(θ¹F¹X¹X1, θ¹G¹X¹X1)
```

Fig. 2. Decomposition of the typing $\theta \in \mathsf{Fun}[\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}],\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]]$. The numbers on the right indicate the inference rules that were used.

Among the declared parameters, there is no object of the category E, so the heuristic looks for parameters whose types contain the goal type. It finds the functor F with range E, which reduces the task of constructing an object of E to constructing an object $z : \mathsf{C}\times\mathsf{D}$ and applying F^1 to it. Since objects in $\mathsf{C}\times\mathsf{D}$ are pairs (A, X) where $A : \mathsf{C}$ and $X : \mathsf{D}$, the task is now finding an object in C and one in D. There are obvious choices for these two objects in the type environment, which means that all the components of the term have been identified. As a result, the heuristic returns the specification $\theta^1 F^1 A^1 X = F^1(A, X)$.

Determining the arguments of a functor or natural transformation is not always as straightforward. In the above case, the parameters in the type environment could be taken directly as components of the term because their types fit the requirements on these components. In other cases, the type environment may provide only an object where an arrow is needed or vice versa. In these situations the most obvious choice is turning an object into an identity arrow and an arrow into its domain or codomain, depending on the typing requirements.

To solve subgoal 1.1.2. we have to build an arrow in $\mathsf{E}(\theta^1 F^1 A^1 X, \theta^1 F^1 A^1 Y)$ from the components $F : \mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$, $A : \mathsf{C}$, $X, Y : \mathsf{D}$, and $k : \mathsf{D}(X, Y)$. Since subgoal 1.1.1. has already been solved, the equality $\theta^1 F^1 A^1 X = F^1(A, X)$ can be used to simplify the goal type to $\mathsf{E}(F^1(A, X), F^1(A, Y))$. To build an arrow of that type from the given parameters, the heuristic has to apply F^2 to an arrow $h \in \mathsf{C}\times\mathsf{D}((A, X), (A, Y))$, i.e. to a pair of arrows (f, g) where $f : \mathsf{C}(A, A)$ and $g : \mathsf{D}(X, Y)$. For the latter, we can pick k but there is no immediate match for $f : \mathsf{C}(A, A)$. Since $\theta^1 F^1 A^2\, k$, the component of θ that shall be specified in this step, explicitly mentions A, the only choice for an arrow in $\mathsf{C}(A, A)$ is the identity 1_A. As a result, the heuristic returns the specification $\theta^1 F^1 A^2\, k = F^2(1_A, k)$.

Subgoal 1.2.1. can be solved in the same manner, which leads to the specification $\theta^1 F^2\, f\, X = F^2(f, 1_X)$. The solution of subgoal 2.1.1. proceeds as the one for subgoal 1.1.1. and yields $\theta^2 \varphi X X_1 = \varphi(X, X_1)$.

In some cases, parameters and identities alone are not sufficient to satisfy a typing conditions, but a simple composition of natural transformation and functor in the style of the equality rule (17) would do so. In this case, the heuristic has to use the functor and its arguments twice in different ways. Although this solution is less obvious it is still considered a standard pattern of reasoning.

We have developed a *calculus for witness construction* that formalizes the heuristic described above. Rules decompose a goal sequent similar to our proof rules in section 3.2 and come with a mechanism that composes sets of specification equations for subgoal sequents into specification equations for the main goal.

There are a few rules that construct specification equations from scratch. If an element z of type Δ has been declared then it can be used as witness to satisfy the type judgment $x \in \Delta$, i.e. we construct the specification equation $x = z$. Furthermore, if the declaration contains a type $\Delta[V_1, ..V_n]$ with free type variables V_i and the type judgment contains an instantiated version $\Delta[T_1, ..T_n]$, then the equations $V_1 = T_1, .., V_n = T_n$ will be constructed as well. In our calculus we write this rule as follows

$$\Gamma,\ z{:}\Delta[V_1, ..V_n] \vdash x \in \Delta[T_1, ..T_n] \qquad \text{specs } \{x = z, V_1 = T_1, .., V_n = T_n\},$$

where the notation specs EQ indicates that the set of specification equations EQ will be constructed if the rule can be applied successfully.

There is also a rule that constructs identities to satisfy a type judgment $f \in \mathsf{C}(A, A)$ if the type environment contains a declaration of an object A of C and a rule that constructs domains or codomains to satisfy a type judgment $x \in \mathsf{C}$ if the type environment contains a declaration of an arrow $f \in \mathsf{C}(A, B)$.

Other rules decompose category constructors like functors or products that occur in the type environment or in the type judgment. For instance, in order to use a functor $F{:}Fun[\mathsf{C},\mathsf{D}]$ when constructing a term $x \in \Delta$ one has to construct an object z of C and show how to use an object y of D in the construction of x. If both goals succeed and yield specification equations EQ_1 and EQ_2 then the specification equation for the main goal is the union of EQ_1 and EQ_2 where y is being replaced by $F^1 z$.

$$\begin{array}{ll} \Gamma,\ F{:}Fun[\mathsf{C},\mathsf{D}] \vdash x \in \Delta & \text{specs } EQ_1 \cup EQ_2[F^1 z/y] \\ \quad \Gamma \vdash z \in \mathsf{C} & \text{specs } EQ_1 \\ \quad \Gamma,\ y : \mathsf{D} \vdash x \in \Delta & \text{specs } EQ_2 \end{array}$$

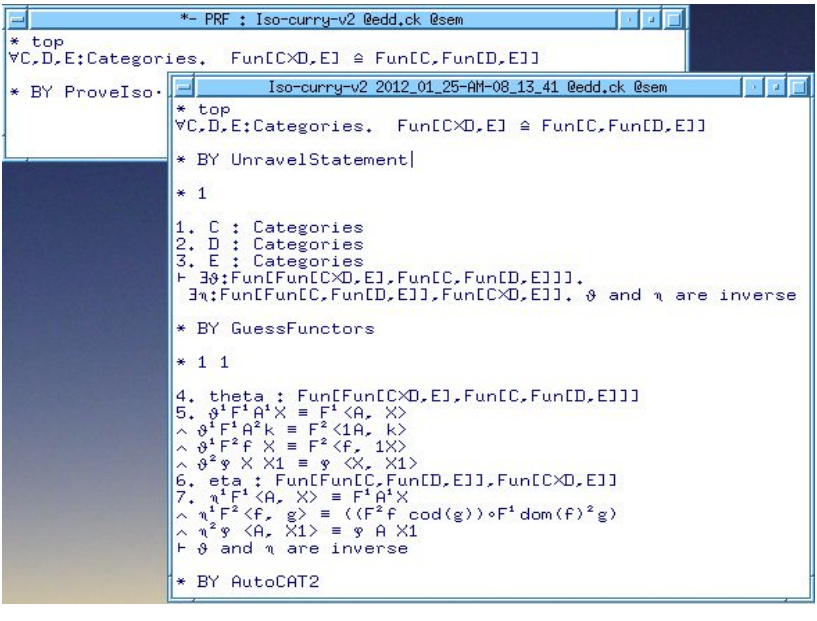
For each constructor there are two rules for decomposing objects and arrows in a type judgment and two rules for decomposing objects and arrows in the type environment. There are also equality reduction rules that simplify a type judgment or a part of the environment by applying a known equality. Figure 3 shows the fragment of our calculus that is necessary for dealing with sequents containing functors and products.

1. Basic rules:
   - $\Gamma,\ z{:}\Delta[V_1,..V_n] \vdash x \in \Delta[T_1,..T_n]$ specs $\{x = z, V_1 = T_1, .., V_n := T_n\}$
   - $\Gamma,\ A{:}\mathcal{C} \vdash f \in \mathcal{C}(A,A)$ specs $\{f = 1_A\}$
   - $\Gamma,\ f{:}\mathcal{C}(A,B) \vdash x \in \mathcal{C}$ specs $\{x = dom(f)\}$
     $\Gamma,\ f{:}\mathcal{C}(A,B) \vdash x \in \mathcal{C}$ specs $\{x := cod(f)\}$
   - $\Gamma,\ f{:}\mathcal{C}(A,B) \vdash h \in \mathcal{C}(A,X)$ specs $\{h = g \circ f\} \cup EQ$
     $\Gamma,\ f{:}\mathcal{C}(A,B) \vdash g \in \mathcal{C}(B,X)$ specs EQ
     $\Gamma,\ g{:}\mathcal{C}(B,X) \vdash h \in \mathcal{C}(A,X)$ specs $\{h = g \circ f\} \cup EQ$
     $\Gamma,\ g{:}\mathcal{C}(B,X) \vdash f \in \mathcal{C}(A,B)$ specs EQ
   - $\Gamma,\ exp = t \vdash x \in \Delta$ specs EQ
     $\Gamma,\ exp = t \vdash x \in \Delta[t/exp]$ specs EQ
     $\Gamma,\ exp = t,\ y{:}\Gamma' \vdash x \in \Delta$ specs EQ
     $\Gamma,\ exp = t,\ y{:}\Gamma'[t/exp] \vdash x \in \Delta$ specs EQ
2. Functors:
   - $\Gamma \vdash F \in Fun[\mathcal{C},\mathcal{D}]$ specs $EQ_1 \cup EQ_2$
     $\Gamma,\ c : \mathcal{C} \vdash F^1 c{:}\mathcal{D}$ specs EQ_1
     $\Gamma,\ EQ_1,\ f : \mathcal{C}(A,B) \vdash F^2 f \in \mathcal{D}(F^1 A, F^1 B)$ specs EQ_2
   - $\Gamma \vdash \varphi \in Fun[\mathcal{C},\mathcal{D}](F,G)$ specs EQ
     $\Gamma,\ A : \mathcal{C} \vdash \varphi A \in \mathcal{D}(F^1 A, G^1 A)$ specs EQ
   - $\Gamma,\ F{:}Fun[\mathcal{C},\mathcal{D}] \vdash x \in \Delta$ specs $EQ_1 \cup EQ_2[F^1 z/y]$
     $\Gamma \vdash z \in \mathcal{C}$ specs EQ_1
     $\Gamma,\ y : \mathcal{D} \vdash x \in \Delta$ specs EQ_2
     $\Gamma,\ F{:}Fun[\mathcal{C},\mathcal{D}] \vdash x \in \Delta$ specs $EQ_1 \cup EQ_2[F^2 f/h]$
     $\Gamma \vdash f \in \mathcal{C}(T_1,T_2)$ specs EQ_1
     $\Gamma,\ A,B : \mathcal{C},\ h : \mathcal{D}(F^1 A, F^1 B) \vdash x \in \Delta$ specs $EQ_2 \cup \{A = T_1, B = T_2\}$
   - $\Gamma,\ \varphi{:}Fun[\mathcal{C},\mathcal{D}](F,G) \vdash x \in \Delta$ specs $EQ_1 \cup EQ_2[\varphi A/h]$
     $\Gamma \vdash A \in \mathcal{C}$ specs EQ_1
     $\Gamma,\ h{:}\mathcal{D}(F^1 A, G^1 A) \vdash x \in \Delta$ specs EQ_2
3. Products:
   - $\Gamma \vdash z \in \mathcal{C} \times \mathcal{D}$ specs $EQ_1 \cup EQ_2 \cup \{z = \langle c,d \rangle\}$
     $\Gamma \vdash c \in \mathcal{C}$ specs EQ_1
     $\Gamma \vdash d \in \mathcal{D}$ specs EQ_2
   - $\Gamma \vdash f \in \mathcal{C} \times \mathcal{D}(\langle A,X \rangle, \langle B,Y \rangle)$ specs $EQ_1 \cup EQ_2 \cup \{f = \langle g,h \rangle\}$
     $\Gamma \vdash g \in \mathcal{C}(A,B)$ specs EQ_1
     $\Gamma \vdash h \in \mathcal{D}(X,Y)$ specs EQ_2
   - $\Gamma,\ z{:}\mathcal{C} \times \mathcal{D} \vdash x \in \Delta$ specs $EQ \cup \{z = \langle c,d \rangle\}$
     $\Gamma,\ c{:}\mathcal{C},\ d{:}\mathcal{D} \vdash x \in \Delta$ specs EQ
   - $\Gamma,\ f{:}\mathcal{C} \times \mathcal{D}(\langle A,X \rangle, \langle B,Y \rangle) \vdash x \in \Delta$ specs $EQ \cup \{f = \langle g,h \rangle\}$
     $\Gamma,\ g{:}\mathcal{C}(A,B),\ h{:}\mathcal{D}(X,Y) \vdash x \in \Delta$ specs EQ

Fig. 3. Calculus for witness construction

To find a set of first-order specification equations that satisfies a given type judgment, our witness construction strategy iteratively applies rules of the calculus of witness construction until specification equations can be constructed and then composes the specification equations at the leave nodes of the decomposition tree into specification equations for the main goal. If more than one rule can be applied, it applies them in a predefined order of "simplicity". If a rule generates more than one subgoal sequent, then the solution for the first subgoal may be used while solving the second.

We have integrated this strategy into a proof tactic `ProveIso` for proving isomorphisms between categories whose proofs are considered trivial by mathematicians. `ProveIso` first unfolds the definition of isomorphisms and decomposes the proof goal. Afterwards the witness construction strategy will guess values for the functors θ and η between the two categories and finally the automated proof search procedure `AutoCAT2` will be called to validate that θ and η are indeed functors of the appropriate types and that they are inverse to each other.



We have applied this tactic to a small collection of isomorphism problems involving functor categories, product categories, and opposite categories. In each case, the isomorphism could be proven by `ProveIso` without a need for further interaction with the user. The screenshot above shows the formal proof that the functor categories $\mathsf{Fun[C\times D, E]}$ and $\mathsf{Fun[C, Fun[D, E]]}$ are isomorphic. On the top-level of this proof a user will only see that the proof was successful (indicated by a star in front of the tactic call `BY ProveIso`). Most users will be satisfied with that amount of information. For users interested in details of the proof the Nuprl system can display the proof tree in several layers of abstraction. The first layer,

shown in the snapshot as well, reveals the key idea that was necessary to solve the problem, but hides the tedious details involved in validating the solution.

Users interested in even more details may subsequently unfold the complete proof tree. However, one should be aware that this tree is huge. It takes 1046 and, respectively, 875 basic inferences to prove that θ and η are indeed functors of the appropriate types and another 1141 inferences to prove that they are inverse to each other. The overall structure of this proof is similar to the hand-constructed one described in [Koz04], which required many hours of careful work to complete. In contrast to that the creation of the proof with `ProveIso` was fully automated and took only a few seconds to complete.

5.2 Proving Categories to Be Naturally Isomorphic

Proving the naturality of an isomorphism between two categories C_1 and C_2 is more demanding than just proving them to be isomorphic since the inverse functors θ and η between C_1 and C_2 now have to be natural transformation of type $\mathsf{Fun}[\mathsf{CAT},\mathsf{Cat}](U,V)$ and $\mathsf{Fun}[\mathsf{CAT},\mathsf{Cat}](V,U)$, where CAT is a yet to be determined product of the large categories Cat and $\mathsf{Cat}^{\mathsf{op}}$ that fit the component categories of C and D and their polarities[3] and U and V are (unknown) elements of $\mathsf{Fun}[\mathsf{CAT},\mathsf{Cat}]$.

Constructing CAT is straightforward. For each component category of C_1 we determine the polarity of its occurrence in C_1, and choose the category Cat if it occurs positively in C_1 (and C_2, respectively) and $\mathsf{Cat}^{\mathsf{op}}$ if it occurs negatively. If the respective polarities are different in C_1 and C_2, then there is no simple natural isomorphism between the two categories and construction fails.[4] For the isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$, for instance, CAT has to be $\mathsf{Cat}^{\mathsf{op}}\times\mathsf{Cat}^{\mathsf{op}}\times\mathsf{Cat}$, since C and D occur with negative polarities in $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ while E occurs positively.

Finding the functors U and V seems more difficult but is still considered obvious because all the relevant information for specifying them is expected to be contained in the terms that describe the construction of C_1 and C_2 from their components. Applying the object component of U and V to a tuple of component categories, for instance, has to result in C_1 and C_2, respectively. Therefore specifications for U^1 and V^1 and also a typing of U^2 and V^2 can be easily constructed and a procedure similar to our witness construction strategy should be able to find a complete specification of all the first-order components of U and V. For the natural isomorphism between $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ this approach gives us the two specifications

$$U^1(\mathsf{C},\mathsf{D},\mathsf{E})=\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}] \quad \text{and} \quad V^1(\mathsf{C},\mathsf{D},\mathsf{E})=\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$$

as well as the typings

[3] A negative polarity indicates that a component category occurs on the left side of a functor. Otherwise the polarity is positive.

[4] We believe that in such situations the two categories are not isomorphic at all.

$U^2(f,g,h) \in \mathsf{Fun}[\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}],\mathsf{Fun}[\mathsf{C}'\times\mathsf{D}',\mathsf{E}']]$ and

$V^2(f,g,h) \in \mathsf{Fun}[\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]],\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]]$.

To solve the latter requirement for U, we have to construct a functor between the categories $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C}'\times\mathsf{D}',\mathsf{E}']$ from functors f, g, and g between C and C', D and D', and E and E'. This construction depends only on the construction of the category $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ from its components but not on the fact that $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ have to be isomorphic. Since U shall be the domain of a natural transformation on large categories, one should expect that there is a "natural" method to construct its arrow component in a way that fits the construction of its object component.

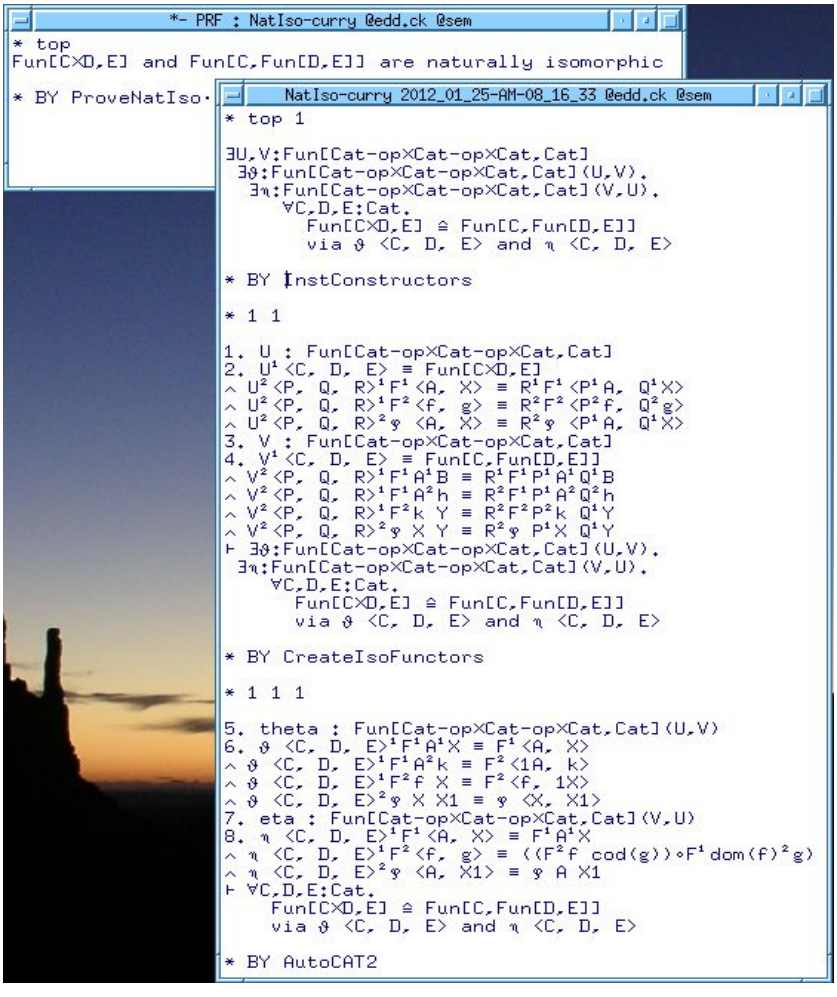
Therefore, our approach to finding specifications that satisfy the requirements on the functors U and V is based on the hypothesis that there should be a natural method to extend a function that constructs a category from component categories into a functor on the category of categories whose object component is the given function. To formalize such a method we have developed a *calculus of constructor functors*. In this calculus we consider constructs like product, coproduct, functor, opposite, empty, or unit categories as object component of a functor on large categories and describe an arrow component that is naturally associated with it. For example, the *product constructor* is described as functor $\mathcal{F}_{\mathsf{Prod}} : \mathsf{Fun}[\mathsf{Cat}\times\mathsf{Cat},\mathsf{Cat}]$, where $\mathcal{F}_{\mathsf{Prod}}{}^1(\mathsf{C},\mathsf{D}) = \mathsf{C}\times\mathsf{D}$ and $\mathcal{F}_{\mathsf{Prod}}{}^2(f,g) = (f,g)$. The *functor constructor* is a functor $\mathcal{F}_{\mathsf{Fun}} : \mathsf{Fun}[\mathsf{Cat}\times\mathsf{Cat},\mathsf{Cat}]$, where $\mathcal{F}_{\mathsf{Fun}}{}^1(\mathsf{C},\mathsf{D}) = \mathsf{Fun}[\mathsf{C},\mathsf{D}]$ and $\mathcal{F}_{\mathsf{Fun}}{}^2(f,g)^1(F) = g{\circ}F{\circ}f$ $\mathcal{F}_{\mathsf{Fun}}{}^2(f,g)^2(\varphi) = g^2{\circ}\varphi{\circ}f^1$.

Using the specifications of elementary constructor functors a constructor for a given category is constructed by decomposing the category into basic constructor functions and composing the associated functors accordingly.

We have integrated this strategy into a proof tactic `ProveNatIso` for proving isomorphisms between categories to be natural. Like `ProveIso`, `ProveNatIso` first unfolds and decomposes the definition of natural isomorphisms. Then the domain and codomain U and V of the natural transformations will be constructed using the calculus of constructor functors and afterwards the natural transformations θ and η using the calculus of witness construction. Finally the tactic `AutoCAT2` will validate the required properties on θ and η.

We have applied this tactic to the same collection of isomorphism problems as before and were able to prove all isomorphisms to be natural. The screenshot below shows the formal proof that the functor categories $\mathsf{Fun}[\mathsf{C}\times\mathsf{D},\mathsf{E}]$ and $\mathsf{Fun}[\mathsf{C},\mathsf{Fun}[\mathsf{D},\mathsf{E}]]$ are naturally isomorphic as well as the first layer of the proof tree that has been constructed by `ProveNatIso`.

Again, the overall structure of this proof is similar to the hand-constructed one described in [Koz04], which indicates that the strategy does indeed automate the most obvious line of reasoning. Furthermore, the fact that the proofs of all "trivial" isomorphisms could be proven without a need for further interaction with the user shows that these proofs are in fact trivial in the sense that only natural constructions, standard forms of reasoning, and meticulous attention to detail are required to solve the problem.



6 Conclusion and Future Work

We have presented an implementation of Kozen's axiomatization of elementary category theory [Koz04] using the Nuprl logical framework [AC+00, AB+05] and a collection of proof tactics that automate standard patterns of reasoning in logic and category theory. We have demonstrated the effectiveness of this approach by automatically deriving proofs of natural isomorphisms between categories, one example of which is presented in detail above. The system works very well on the examples we have tried.

There is a number of alternative approaches to a formalization and automation of category theory. The Mizar approach [Miz, Try92, Ba01a, Ba01b, Ba01c] aims at a formal reconstruction of mathematical knowledge in a computer-oriented environment.Mizar's library seems to contain the most comprehensive collection

of theorems but it does not provide a mechanism for automating domain-specific reasoning tasks.

The system of Cáccamo and Winskel [CW01] presents a second order calculus for a fragment of category theory, which permits a more elegant representation of higher-order constructs like functors or natural transformations. Since higher-order reasoning is more difficult to automate, however, facts like Yoneda's lemma need to be stated as a rules instead of being derived as theorems.

Burstall and Rydeheard [RB88] have implemented a substantial fragment of *Computational Category Theory* in Standard ML. The focus of their work, however, was not on the development of proofs but on creating a basis for the use of category theory in program design.

Other approaches aim at formalizations of category theory in interactive proof systems. Glimming [Gli01] describes a development of basic category theory and a couple of concrete categories (Unit, Set, Gal, Cat, Poset and Cpo) in Isabelle/HOL. O'Keefe [O'K04] presents a formalization of category theory in Isabelle that focuses on the readability of proofs, aiming at a representation close to one in a mathematical textbook. In both approaches there are no attempts to improve automation beyond Isabelle's generic prover.

Dyckhoff [Dyc08] presents an formalization of category theory in Martin-Löf type theory that has some similarity to Kozen's first-order axiomatization but uses higher-order constructs in some of the rules. Dyckhoff hints at techniques to automate reasoning in his calculus but there is no actual implementation.

In the Coq library there are two contributions concerning category theory. The development of Saïbi and Huet [HS95, Sai95] contains definitions and constructions up to cartesian closed categories, which are then applied to the category of sets. The formalization is directly based on the Coq's type theory. In contrast to that Simpson's formalization [Sim04] is set up in a ZFC-like environment and includes some tactics to improve the automation. Both approaches, however, are not described in any official publication.

A key difference between these works and our approach is that we have given a full implementation of an independent calculus for reasoning about category. In addition, we have provided a family of tactics that allow many proofs to be automated. None of the other implementations we have encountered make any attempt to isolate an independent formal axiomatization of the elementary theory. Instead, they embed category theory into some other logic, and reasoning relies mostly on the underlying logic.

There are a number of technical insights that we have observed in the course of this work, which show the advantage of using the Nuprl system as framework for implementing category theory.

- The use of abstractions and display forms is crucial for comprehensibility. It is often very difficult to keep track of typing judgments currently in force. Judicious choice of the display form can make a great difference in readability.
- The combination of rule objects and rule compiler are essential for a faithful implementation of proof calculi. Rule objects contain visual representations of proof rules that look almost identical to the version on paper and can

easily be checked for correctness while the actual implementation of the rule is being generated by the rule compiler.
- Formal proofs, even of elementary facts, have thousands of basic inferences, which are often quite tedious and do not lend much insight. This indicates to us that elementary category theory is a very good candidate for automation.
- The ability to inspect proof objects at increasing levels of abstraction makes it possible to generate proofs that humans can understand and check to the very last detail even if the formal proof is extremely large.
- Nuprl's tactic mechanism makes it possible to quickly implement and test new reasoning strategies and to embed new reasoning patterns when they are discovered in the course of a not yet fully automated proof.
- Reasoning in elementary category theory can be automated very well once there is an understanding of the typical kinds of reasoning that mathematicians consider obvious. As most reasoning steps are based on straightforward decomposition and directed rewriting for equations, proof strategies spend most of their time building the proof. Apart from guessing witnesses, which involves investigating a small set of alternative choices, there is virtually no backtracking involved and the bulk of the development is completely deterministic, being driven by typing considerations.
- Proofs that are considered trivial from an intellectual point of view are in fact trivial in the sense that a computer program can find them without having to rely on sophisticated heuristics.

For the future, we plan to gain more experience by attempting to automate more of the basic theory. We need more experience with the different types of arguments that arise in category theory so that we will be better able to automate proofs that require witnesses for existential quantifiers. We believe that our *calculus for witness construction* will be useful beyond proofs of isomorphisms, as the most obvious solution for a problem in category theory is often the "simplest" element of the given type, which is exactly what the strategy generates. In the same way our *calculus of constructor functors* should be useful beyond proofs of natural isomorphisms, as it provides functors and natural transformations that come naturally with a given category.

To improve both the efficiency of a proof search and the readability of the constructed proofs we also plan to introduce higher levels of reasoning that compose theorems about general category-theoretical arguments instead of only applying basic inference rules. This form of compositional reasoning has proven successful in the formal optimization of communication systems [LK+99], where we could reduce a huge amount of basic inference steps to a proof that could be constructed in a few seconds.

Finally, we would like to mention an intriguing theoretical open problem. The proofs of natural isomorphisms between two categories C_1 and C_2 that we have described break down into two parts. The first part argues that C_1 and C_2 are isomorphic, and the second part argues that the isomorphism is natural. As Mac Lane describes it [McL71, p. 2], *naturality*, applied to a parameterized construction, says that the construction is carried out "in the same way" for all

instantiations of the parameters. Of course, there is a formal definition of the concept of naturality in category theory itself, and it involves re-parameterizing the result in terms of functors in place of objects, natural transformations in place of arrows. But any constructions in the formal proof π of the first part of the theorem, just the isomorphism of the two parameterized functor categories, would work "in the same way" for all instantiations of the parameters, by virtue of the fact that the formal proof π is similarly parameterized. In fact, our proof strategy based on the calculus of constructor functors has attempted exactly that and succeeded in proving an isomorphism to be natural in each case where we could prove two categories to be isomorphic.

This leads us to ask: Under what conditions can one extract a proof of naturality *automatically* from π? That is, under what conditions can a proof in our formal system be automatically retooled to additionally establish the naturality of the constructions involved? Extracting naturality in this way would be somewhat analogous to the extraction of programs from proofs according to the Curry–Howard isomorphism. We believe that extracting naturality is possible at least for categories that can be described in terms of constructor functors, as this leads immediately to the domains and codomains of the natural transformations θ and η between the two categories while θ and η are constructed in the same way as in the proof the isomorphism between C_1 and C_2. But a formal proof of this conjecture still needs to be given.

Acknowledgements. We thank Dexter Kozen for introducing us to the topic of implementing and automating elementary category theory and for explaining the informal reasoning patterns behind the proof in [Koz04].

References

[AB+05] Allen, S., Bickford, M., Constable, R., Eaton, R., Kreitz, C., Lorigo, L., Moran, E.: Innovations in computational type theory using Nuprl. Journal of Applied Logic 4(4), 428–469 (2006)

[AC+00] Allen, S., Constable, R., Eaton, R., Kreitz, C., Lorigo, L.: The Nuprl Open Logical Environment. In: McAllester, D. (ed.) CADE 2000. LNCS, vol. 1831, pp. 428–469. Springer, Heidelberg (2000)

[Ba01a] Bancerek, G.: Concrete categories. Journal of Formalized Mathematics 13 (2001)

[Ba01b] Bancerek, G.: Miscellaneous facts about functors. Journal of Formalized Mathematics 13 (2001)

[Ba01c] Bancerek, G.: Categorial background for duality theory. Journal of Formalized Mathematics 13 (2001)

[BW90] Barr, M., Wells, C.: Category Theory for Computing Science. Prentice Hall (1990)

[CA+86] Constable, R., et al.: Implementing Mathematics with the Nuprl proof development system. Prentice Hall (1986)

[Con08] Constable, R.: Computational Type Theory. Scholarpedia 4(2), 7618 (2008)

[CW01] Cáccamo, M.J., Winskel, G.: A higher-order calculus for categories. Technical Report RS-01-27, BRICS, University of Aarhus (2001)

[Dyc08] Dyckhoff, R.: Category theory as an extension of Martin-Löf type theory. Research Report CS/85/3, Revised 1988 (1988)

[EM45] Eilenberg, S., MacLane, S.: General theory of natural equivalences. Trans. Amer. Math. Soc. 58, 231–244 (1945)

[Gli01] Glimming, J.: Logic and automation for algebra of programming. Master thesis, University of Oxford (2001)

[HS95] Huet, G., Saïbi, A.: Constructive category theory. In: Joint CLICS-TYPES Workshop on Categories and Type Theory. MIT Press (1995)

[EB70] Knuth, D., Bendix, P.: Simple word problems in universal algebra. In: Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970)

[KHH98] Kreitz, C., Hayden, M., Hickey, J.: A Proof Environment for the Development of Group Communication Systems. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS (LNAI), vol. 1421, pp. 317–331. Springer, Heidelberg (1998)

[Koz04] Kozen, D.: Toward the automation of category theory. Technical Report 2004-1964, Computer Science Department, Cornell University (2004)

[KKR06] Kozen, D., Kreitz, C., Richter, E.: Automating Proofs in Category Theory. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 392–407. Springer, Heidelberg (2006)

[Kre02] Kreitz, C.: The Nuprl Proof Development System, V5: Reference Manual and User's Guide. Computer Science Department, Cornell University (2002)

[Kre04] Kreitz, C.: Building reliable, high-performance networks with the Nuprl proof development system. Journal of Functional Programming 14(1), 21–68 (2004)

[LK+99] Liu, X., Kreitz, C., van Renesse, R., Hickey, J., Hayden, M., Birman, K., Constable, R.: Building reliable, high-performance communication systems from components. In: 17th ACM Symposium on Operating Systems Principles, vol. 34(5), pp. 80–92 (1999); Operating Systems Review

[McL71] MacLane, S.: Categories for the Working Mathematician. Springer (1971)

[Miz] Mizar home page, `http://www.mizar.org`

[ML84] Martin-Löf, P.: Intuitionistic Type Theory, Bibliopolis (1984)

[O'K04] O'Keefe, G.: Towards a readable formalisation of category theory. In: Atkinson, M. (ed.) Computing: The Australasian Theory Symposium. ENTCS, vol. 91, pp. 212–228. Elsevier (2004)

[RB88] Rydeheard, D., Burstall, R.: Computational Category Theory. International Series in Computer Science. Prentice Hall (1988)

[RRW91] Reed, G.M., Roscoe, A.W., Wachter, R.F.: Topology and Category Theory in Computer Science. Oxford University Press (1991)

[Sai95] Saïbi, A.: Constructive category theory (1995), `http://coq.inria.fr/contribs/category.tar.gz`

[Sim04] Simpson, C.: Category theory in ZFC (2004), `http://coq.inria.fr/contribs/CatsInZFC.tar.gz`

[Try92] Trybulec, A.: Some isomorphisms between functor categories. Journal of Formalized Mathematics 4 (1992)

On the Final Coalgebra of Automatic Sequences

Clemens Kupke[1] and Jan J.M.M. Rutten[2,3]

[1] University of Oxford
[2] Centrum Wiskunde & Informatica (CWI)
[3] Radboud University Nijmegen

Abstract. Streams are omnipresent in both mathematics and theoretical computer science. Automatic sequences form a particularly interesting class of streams that live in both worlds at the same time: they are defined in terms of finite automata, which are basic computational structures in computer science; and they appear in mathematics in many different ways, for instance in number theory. Examples of automatic sequences include the celebrated Thue-Morse sequence and the Rudin-Shapiro sequence. In this paper, we apply the coalgebraic perspective on streams to automatic sequences. We show that the set of automatic sequences carries a final coalgebra structure, consisting of the operations of head, even, and odd. This will allow us to show that automatic sequences are to (general) streams what rational languages are to (arbitrary) languages.

With all our best wishes to Dexter Kozen, on the occasion of his 60th birthday.

1 Introduction

The set of infinite sequences, or *streams*, over an alphabet A is defined by

$$A^\omega = \{\sigma \mid \sigma : \mathbb{N} \to A\}$$

In universal coalgebra [Rut00], which is a general theory of the behaviour of dynamical systems and infinite data types, the set of streams is the prototypical example of a *final coalgebra* much in the same way as in universal algebra the set of natural numbers is the typical example of an initial algebra. The (final) coalgebra structure of A^ω is given by the isomorphism

$$A^\omega \to A \times A^\omega \qquad \sigma \mapsto (\sigma(0),\, \sigma')$$

which maps a stream σ to the pair consisting of its *head* or *initial value* $\sigma(0)$, and its *tail* or *stream derivative* σ', which is defined by

$$\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$$

This elementary structure on A^ω gives rise to a surprisingly powerful collection of definition and proof methods that are based on the finality of the stream

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 149–164, 2012.

coalgebra $(A^\omega, (\_)(0), (\_)')$, ie, based on the fact that from any other coalgebra of the same type there is exactly one coalgebraic morphism into the stream coalgebra.

For instance, in so-called *stream calculus* [Rut05], one defines streams by means of *stream differential equations*, in close analogy to classical calculus in mathematics. Examples are

$$\sigma' = 3 \times \sigma \qquad \sigma(0) = 1$$

(with $3 \times \sigma = (3 \cdot \sigma(0), 3 \cdot \sigma(1), 3 \cdot \sigma(2), \ldots)$), which can be easily seen to define the stream $(3^0, 3^1, 3^2, 3^3, \ldots)$; and

$$\sigma' = \sigma \times \sigma \qquad \sigma(0) = 1$$

(where in this case $\times$ stands for convolution product), which defines the stream $(1, 1, 2, 5, 14, \ldots)$ of the so-called Catalan numbers. Furthermore, stream calculus comes along with a proof principle called *coinduction*, by which two streams can be shown to be equal by the construction of a suitable $\{\mathsf{head}, \mathsf{tail}\}$-*bisimulation relation*.

Streams are omnipresent in both mathematics and theoretical computer science. *Automatic sequences* [Fog02, AS03] form a particularly interesting class of streams that live in both worlds at the same time: they are defined in terms of finite automata, which are basic computational structures in computer science; and they appear in mathematics in many different ways, for instance in number theory. We will see a formal definition later but, in a nutshell, a stream is (2-)automatic if its n-th value is obtained by feeding the binary encoding of the number n into a Moore machine, and reading off the output value of the state thus reached. Examples of automatic sequences include the celebrated Thue-Morse sequence and the Rudin-Shapiro sequence.

In this paper, we set out to apply the coalgebraic perspective on streams to automatic sequences. As it happens, we shall not be using the aforementioned stream calculus, which is based on the standard final coalgebra structure of head and tail that we saw above. Instead, we will use a stream calculus that could be called *non-standard* because it is based on a different coalgebra structure on streams, which was recently introduced in [KR10]. This coalgebra structure is defined by

$$\langle \mathsf{head}, (\mathsf{even}, \mathsf{odd}) \rangle : A^\omega \to A \times (A^\omega)^2$$

where $\mathsf{head}(\sigma) = \sigma(0) \in A$ is again the initial value of σ and where

$$\mathsf{even}(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots) \qquad \mathsf{odd}(\sigma) = (\sigma(1), \sigma(3), \sigma(5), \ldots)$$

As we shall see, the coalgebra $(A^\omega, \langle \mathsf{head}, (\mathsf{even}, \mathsf{odd}) \rangle)$ is final among (a subclass of) the coalgebras of type $S \to A \times S^2$. Such coalgebras are known in the literature as 2-automata (with outputs in A); they are also called Moore machines.

The above final coalgebra structure on A^ω gives rise to a new type of stream calculus, in which streams can be defined by so-called $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations. For instance,

$$\mathsf{head}(\sigma) = 0 \qquad \mathsf{even}(\sigma) = \mathsf{zeros} \qquad \mathsf{odd}(\sigma) = \mathsf{ones}$$

with $\mathsf{zeros} = (0, 0, 0, \ldots)$ and $\mathsf{ones} = (1, 1, 1, \ldots)$ defines the stream

$$\sigma = (0, 1, 0, 1, 0, 1, \ldots).$$

Another, less trivial example is

$$\mathsf{head}(M) = 0 \qquad \mathsf{even}(M) = M \qquad \mathsf{odd}(M) = N$$

$$\mathsf{head}(N) = 1 \qquad \mathsf{even}(N) = N \qquad \mathsf{odd}(N) = M$$

which is a system of two equations that has the Thue-Morse sequence M (and its complement N) as its unique solution. The Thue-Morse sequence has the property that $M(i) = 0$ iff the number of 1's in the binary representation of i is even and that $M(i) = 1$ otherwise, ie, $M = 01101001\ldots$ - for more detailed information about this sequence and its properties the reader is referred to [AS03].

As we shall demonstrate, the relevance of the above non-standard stream calculus for the theory of automatic sequences lies in the following observations:

- the definition of automatic sequences by finite automata becomes a universal construction, by the finality of A^ω mentioned above;
- a stream σ is automatic iff the subcoalgebra $[\sigma]$ of the final coalgebra A^ω, obtained by repeatedly applying the operations of odd and even, is finite (much in the same way as a formal language L is regular iff the subcoalgebra it generates in the final coalgebra 2^{A^*} of all formal languages is finite);
- the size of this generated subcoalgebra gives us a well-defined notion of minimal representation for automatic sequences;
- the set of all automatic sequences is final among (a subclass of) all finitely-generated 2-automata;
- a sequence is automatic if and only if it can be defined by a finite system of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations;
- equality of automatic sequences can be proved by coinduction, employing $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-bisimulation relations;
- a more liberal use of non-standard stream differential equations leads to potentially interesting extensions of the set of automatic sequences, similar to the way in which context-free languages extend regular languages.

2 Streams and Automata

Let A be an arbitrary set and let $2 = \{0, 1\}$. A 2-*automaton* with outputs in A is a pair $(S, \langle o, n\rangle)$ consisting of a (finite or infinite) set S of states and a pair of functions

$$\langle o, n\rangle : S \to A \times S^2$$

assigning to each state $s \in S$ its output $o(s) \in A$ and, for each input $i \in 2$, a next state $n(s)(i) \in S$. (Equivalently, 2-automata are coalgebras of the set-functor $F(X) = A \times X^2$.)

Note that every state s has two successor states, $n(s)(0)$ and $n(s)(1)$, which we shall sometimes call the 0-derivative and the 1-derivative of s, and which we often denote by

$$s_0 = n(s)(0) \qquad s_1 = n(s)(1)$$

The notion of derivative can be generalised to binary words as usual.

Definition 1. *For a state $s \in S$ in a 2-automaton $(S, \langle o, n\rangle)$ and for $w \in 2^*$, we define the w-derivative s_w of s by*

$$s_\epsilon = s \qquad s_{w\cdot b} = n(s_w)(b)$$

where ϵ is the empty word and $b \in 2$. □

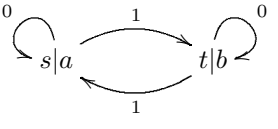
In other words, the w-derivative s_w of some automaton state s is the state that the automaton reaches when starting at state s and reading word w. As usual, automata can be conveniently represented by pictures, in which we denote outputs by

$$s|a \iff o(s) = a$$

and successor states by labeled arrows:

$$s \xrightarrow{0} s_0 \qquad s \xrightarrow{1} s_1$$

Using these conventions, here is an example of a 2-automaton with outputs in $A = \{a, b\}$:



This picture represents an automaton with $S = \{s, t\}$; with outputs $o(s) = a$ and $o(t) = b$; and with derivatives (that is, successor states) $s_0 = s = t_1$ and $s_1 = t = t_0$.

As we already saw in the introduction, the set of streams over A forms a 2-automaton

$$\langle \mathsf{head}, (\mathsf{even}, \mathsf{odd})\rangle : A^\omega \to A \times (A^\omega)^2 \tag{1}$$

given by $\mathsf{head}(\sigma) = \sigma(0) \in A$ and by

$$\mathsf{even}(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots) \qquad \mathsf{odd}(\sigma) = (\sigma(1), \sigma(3), \sigma(5), \ldots)$$

In line with the conventions for 2-automata introduced above, we shall often write

$$\sigma_0 = \mathsf{even}(\sigma) \qquad \sigma_1 = \mathsf{odd}(\sigma)$$

Applying Definition 1 to the automaton of streams then gives us word derivatives of streams, which will play a crucial role later. For instance,

$$\sigma_{011} = \mathsf{odd} \circ \mathsf{odd} \circ \mathsf{even}(\sigma) = (\sigma(6), \sigma(14), \sigma(22), \ldots)$$

Because $\mathsf{even}(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots)$, the output values of σ and its 0-derivative σ_0 are always the same:

$$\mathsf{head}(\sigma) = \sigma(0) = \mathsf{head}(\mathsf{even}(\sigma)) = \mathsf{head}(\sigma_0)$$

Definition 2 (zero-consistency). *More generally, a* 2*-automaton* $(S, \langle o, n\rangle)$ *is called* zero-consistent *if* $o(n(s)(0)) = o(s)$. □

For another interesting (but not zero-consistent) 2-automaton, we consider the set

$$A^{2^*} = \{L \mid L : 2^* \to A\}$$

of all A-weighted languages over 2, together with output and transition functions

$$\langle o, n\rangle : A^{2^*} \to A \times (A^{2^*})^2 \tag{2}$$

given, for any $L : 2^* \to A$, by $o(L) = L(\epsilon)$ and by $n(L)(i) = L_i$ with

$$L_i(w) = L(i \cdot w)$$

for $i \in 2$ and $w \in 2^*$. (As an aside, we note that A-weighted languages over 2 can also be viewed as binary trees with (node) labels in A.)

3 Finality

We shall show that the automaton of all A-weighted languages is final in the family (the category) of all 2-automata. This observation is an instance of a well-known, more general characterisation of the final coalgebra for (Mealy and) Moore machines with arbitrary inputs and outputs. Next we will show that the automaton of all streams is final in the family (the subcategory) of all *zero-consistent* 2-automata. This fact is much less well-known and was recently proved in [KR10], building on earlier work by Rosu [Ros00]. The main contribution of the present paper, in Section 4, will be a new definition and characterisation of automatic sequences, based on the finality of the 2-automaton of streams.

We begin by recalling some basic notions from universal coalgebra [Rut00]. A *homomorphism* from a 2-automaton $(S, \langle o_S, t_S\rangle)$ to a 2-automaton $(T, \langle o_T, n_T\rangle)$ is a function $f : S \to T$ such that for all $s \in S$,

$$o_T(f(s)) = o_S(s) \;\text{ and }\; f(s)_0 = f(s_0), \quad f(s)_1 = f(s_1)$$

Equivalently, f is a homomorphism if it makes the following diagram commute:

$$\begin{array}{ccc} S & \xrightarrow{\;f\;} & T \\ {\scriptstyle \langle o_S, n_S\rangle}\downarrow & & \downarrow{\scriptstyle \langle o_T, n_T\rangle} \\ A \times S^2 & \xrightarrow[1_A \times f^2]{} & A \times T^2 \end{array}$$

Here 1_A is the identity map on A and $f^2 : S^2 \to T^2$ sends (s_0, s_1) to $(f(s_0), f(s_1))$.

Using the notion of homomorphism, we can formulate the following universal property of the automaton of languages.

Theorem 3 (finality of weighted languages). *The* 2*-automaton of A-weighted languages defined in equation (2) is final in the family of* all 2*-automata. That is, for every* 2*-automaton* $(S, \langle o_S, n_S\rangle)$ *there exists a unique homomorphism into the automaton* $(A^{2^*}, \langle o, n\rangle)$*:*

$$\begin{array}{ccc} S & \overset{\exists!\, l}{- - \to} & A^{2^*} \\ {\scriptstyle \langle o_S, n_S\rangle}\downarrow & & \downarrow{\scriptstyle \langle o, n\rangle} \\ A \times S^2 & \xrightarrow[1_A \times l^2]{} & A \times (A^{2^*})^2 \end{array}$$

Proof. We define $l : S \to A^{2^*}$, for $s \in S$ and $w \in 2^*$, by

$$l(s)(w) = o_S(s_w)$$

One can easily show that this turns l into a homomorphism and that this is the only way to do it. □

Next we turn to the automaton of streams, more important for the purposes of this paper. First we define the binary representation of the natural numbers

$$\mathsf{bin} : \mathbb{N} \to 2^*$$

by $\mathsf{bin}(0) = \epsilon$, the empty word, and further, as usual, by

$$\mathsf{bin}(1) = 1 \quad \mathsf{bin}(2) = 01 \quad \mathsf{bin}(3) = 11 \quad \mathsf{bin}(4) = 001$$

and so on (least significant digit first). We have the following useful property.

Lemma 4. *For* $\sigma \in A^\omega$ *and* $n \geq 0$*:* $\sigma(n) = \sigma_{\mathsf{bin}(n)}(0)$*.* □

For instance, $\sigma(6) = \sigma_{\mathsf{bin}(6)}(0) = \sigma_{011}(0) = (\mathsf{odd} \circ \mathsf{odd} \circ \mathsf{even}(\sigma))(0)$.

We just saw that the automaton of all weighted languages is final among all 2-automata. Next we show that the automaton of streams if final among *all zero-consistent* 2-automata.

Theorem 5 (finality of automaton of streams). *For every zero-consistent* 2*-automaton* $(S, \langle o_S, n_S\rangle)$ *there exists a unique homomorphism into the automaton* $(A^\omega, \langle \mathsf{head}, (\mathsf{even}, \mathsf{odd})\rangle)$*:*

$$\begin{array}{ccc} S & \overset{\exists!\, h}{- - \to} & A^\omega \\ {\scriptstyle \langle o_S, n_S\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{head}, (\mathsf{even}, \mathsf{odd})\rangle} \\ A \times S^2 & \xrightarrow[1_A \times h^2]{} & A \times (A^\omega)^2 \end{array}$$

Proof. We define $h : S \to A^\omega$, for $s \in S$ and $n \geq 0$, by

$$h(s)(n) = o_S(s_{\mathsf{bin}(n)})$$

Using Lemma 4, one can show that this is the only possible definition ensuring that h is a homomorphism. □

4 Automatic Sequences

We can now use the finality of the automaton of streams to give our definition of automatic sequence. We need another definition first.

Definition 6 (Streams and Automata). *We say that a stream $\sigma \in A^\omega$ is* generated *by a state s of a (finite or infinite) zero-consistent* 2*-automaton $(S, \langle o, n\rangle)$ if $\sigma = h(s)$, where $h : S \to A^\omega$ is the (by finality unique) homomorphism of Theorem 5.* □

Automatic sequences are precisely those sequences that can be generated by a state of a *finite* automaton, ie, an automaton with a finite number of states.

Definition 7 (automatic). *We call a stream $\sigma \in A^\omega$* automatic *if it is generated by a* finite *zero-consistent* 2*-automaton.* □

We can also characterise automatic sequences in terms of *subautomata*, which we introduce next. For a state $s \in S$ in a 2-automaton $(S, \langle o_S, n_S\rangle)$, we define the *subautomaton induced by s* as the smallest set $[s] \subseteq S$ such that:

$$s \in [s] \quad \text{and} \qquad \forall t \in [s] : \; t_0 \in [s] \quad \text{and} \quad t_1 \in [s]$$

and with outputs and transitions as in S.

Theorem 8 (Automatic Sequences and Subautomata). *For a stream $\sigma \in A^\omega$, the following are equivalent:*

(i) σ is automatic.
(ii) The subautomaton $[\sigma] \subseteq A^\omega$ induced by σ is finite.
(iii) σ has only finitely many derivatives.

Proof. The equivalence of (i) and (ii) follows from elementary universal coalgebra [Rut00]: (ii) implies (i) because the inclusion of a subautomaton into a bigger automaton is always a homomorphism; and (i) implies (ii) because the image of a subautomaton under a homomorphism is always a subautomaton or, more specifically: $h([s]) = [h(s)]$. The equivalence of (ii) and (iii) follows from the definition of subautomaton. □

By this theorem, we have a very precise correspondence between automatic sequences and regular languages, namely:

$$\frac{\text{automatic sequences}}{\text{all streams}} \quad = \quad \frac{\text{regular languages}}{\text{all languages}}$$

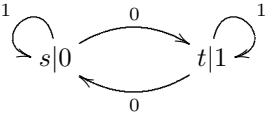
Let us explain. According to Theorem 8 above, a stream σ (over A) is automatic if and only if the subautomaton $[\sigma]$ induced by σ in the final coalgebra A^ω of all streams is finite or, equivalently, if σ has only finitely many derivatives. There is the following similar fact for languages [Con71, Rut98]: a language L (over an alphabet A) is regular if and only if the subautomaton $[L]$ induced by L in the final coalgebra 2^{A^*} of all languages is finite or, equivalently, if L has only finitely many input derivatives.

Discussion

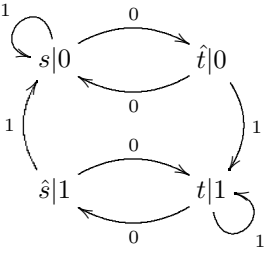
In the remainder of this section, we briefly discuss how our definition of automaticity is related to the existing classical notion(s) in the literature. None of this will play a role in the following sections.

Definition 7 is (almost) equivalent to the following definition [Fog02, page 13]: an infinite sequence (stream) is 2-automatic if it is *generated by a* 2*-automaton in reverse reading* (reading the least significant digit of the binary representation of the natural numbers first). Without spelling out the details of this latter definition, it is almost the same as ours.

We say *almost* because in the classical definition, 2-automata are not required to be zero-consistent. So our present definition might seem more restrictive but it is not. We illustrate this by means of a simple example, omitting the not particularly instructive proof that this holds in general. Let us consider the following example of a 2-automaton with outputs in $A = \{0, 1\}$:



We note that it is not zero-consistent since the output of s is 0 and the output of $s_0 = t$ is 1. However, it can be transformed into the following four state zero-consistent automaton:



which is equivalent to the original automaton in the following sense: One can easily show that the stream generated in reverse reading by this latter automaton (starting in $\hat{s}$) is the same as the stream generated in reverse reading by the original automaton (starting in s).

Then there exists yet another classical definition of automaticity, in which a stream is generated while reading the most significant digit of the binary representation of the natural numbers first. Such a stream is said to be generated in *direct reading*. It has been shown ([Fog02, Proposition 1.3.4.], [AS03, Theorem 5.2.3]) that automaticity in reverse reading and in direct reading are equivalent.

Finally, our Theorem 8 above is also known, be it somewhat implicitly, in the literature, where our (ii) is phrased in terms of 2-kernels; see [Fog02, Proposition 1.3.3.] and [AS03, Theorem 6.6.2] for details.

5 Non-automatic Sequences

Theorem 8 gives us a convenient criterion to decide whether a stream is automatic or not.

For a first example, let us look at the characteristic stream of powers of 2:

$$\tau = (0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, \ldots)$$

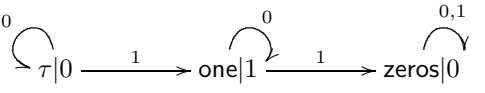
Using elementary stream calculus [Rut05], with $X = (0, 1, 0, 0, \ldots)$, convolution product as multiplication and element-wise addition as sum, we can write τ as

$$\begin{aligned} \tau &= X^{2^0} + X^{2^1} + X^{2^2} + X^{2^3} + \cdots \\ &= X + X^2 + X^4 + X^8 + \cdots \end{aligned}$$

(Equivalently, this can be viewed as a power series representation of τ in the formal variable X.) Computing the (repeated) derivatives of τ gives

$$\tau_0 = \tau \quad \tau_1 = \mathsf{one} \quad \mathsf{one}_0 = \mathsf{one} \quad \mathsf{one}_1 = \mathsf{zeros} \quad \mathsf{zeros}_0 = \mathsf{zeros} = \mathsf{zeros}_1$$

where $\mathsf{one} = (1, 0, 0, \ldots)$ and $\mathsf{zeros} = (0, 0, 0, \ldots)$. Thus we see that τ is generated by the following finite automaton



By Theorem 8, we conclude that τ is automatic.

For a second example, we consider the characteristic sequence of squares

$$\rho = (1, 1, 0, 0, 1, 0, 0, 0, 0, 1, \ldots)$$

or, equivalently,

$$\begin{aligned} \rho &= X^{0^2} + X^{1^2} + X^{2^2} + X^{3^2} + \cdots \\ &= 1 + X + X^4 + X^9 + \cdots \end{aligned}$$

In order to decide whether ρ is automatic or not, we investigate its derivatives. Since

$$\rho_0 = 1 + X^2 + X^8 + X^{18} + X^{32} + X^{50} + X^{72} + \cdots$$

it follows that $\rho_{00} = \rho$ and $\rho_{01} = \mathsf{zeros}$. Next we compute

$$\begin{aligned} \rho_1 &= 1 + X^4 + X^{12} + X^{24} + X^{40} + \cdots \\ \rho_{10} &= 1 + X^2 + X^6 + X^{12} + X^{20} + \cdots \\ \rho_{100} &= 1 + X + X^3 + X^6 + X^{10} + X^{15} + \cdots \end{aligned}$$

where we note that the latter stream is pleasantly regular:

$$\rho_{100} = X^0 + X^{0+1} + X^{0+1+2} + X^{0+1+2+3} + X^{0+1+2+3+4} + X^{0+1+2+3+4+5} + \cdots$$

Next we can easily prove by induction that the first non-zero coefficient of X of the subsequent zero derivatives satisfies, for all $n \geq 1$,

$$\rho_{10^{n+1}} = 1 + X^{2^n - 1} + \cdots$$

Thus all these streams are different. By Theorem 8, we have that ρ is *not* automatic. The reader is invited to compare this proof with that of [AS03, page 167], which is somewhat ad hoc and which is based on the pumping lemma for formal languages.

6 Coinduction (Definitions)

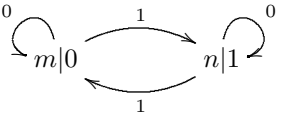
In coalgebra, the property of being final gives rise to both a method for *defining* and a method for *proving* equality (of the elements of the final coalgebra). Such definitions and proofs are often called *coinductive* or: *by coinduction*. In the present section, we shall sketch how coinductive definitions looks like for (automatic) streams. The next section will deal with coinductive proofs.

Coinductive definitions of streams are given by $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations. As an example, we consider the Thue-Morse sequence and its inverse, which we already saw in the introduction. Let the streams $M, N \in 2^\omega$ be defined by the following equations:

$$\begin{array}{lll} \mathsf{head}(M) = 0 & \mathsf{even}(M) = M & \mathsf{odd}(M) = N \\ \mathsf{head}(N) = 1 & \mathsf{even}(N) = N & \mathsf{odd}(N) = M \end{array} \tag{3}$$

We mentioned in the introduction that this system of two differential equations has the Thue-Morse sequence M and its complement N as its unique solution. But how do we know that the system has a solution at all and that this solution is moreover unique?

The affirmative answer is given by finality. We use the differential equations to define a zero-consistent 2-automaton S as follows: let $S = \{m, n\}$ and let outputs (in $A = 2$) and transitions be given by the following picture:



By finality Theorem 5, there exists a unique homomorphism $h : S \to 2^\omega$ which we use to define

$$M = h(m) \quad N = h(n)$$

It is now straightforward to show that the streams M and N are solutions of, that is, satisfy the differential equations above. Computing their first elements gives

$$M = (0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, \ldots)$$
$$N = (1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, \ldots)$$

Moreover, because every pair of solutions of our differential equations will give rise to a homomorphism from S to 2^ω, the unicity of h implies that M and N are the unique solutions. Finally, because S is finite both M and N are automatic.

More generally, we call a finite system of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations *simple* if it is of the following form:

$$\mathsf{head}(x_1) = a_1 \quad \mathsf{even}(x_1) = y_1 \quad \mathsf{odd}(x_1) = z_1$$

$$\cdots$$

$$\mathsf{head}(x_n) = a_n \quad \mathsf{even}(x_n) = y_n \quad \mathsf{odd}(x_n) = z_n$$

where $X = \{x_1, \ldots, x_n\}$ is the set of unknowns, for some $n \geq 1$; and where $a_i \in A$, $y_i \in X$ and $z_i \in X$, for all $1 \leq i \leq n$. We call such a simple system *zero-consistent* if $\mathsf{head}(y_i) = \mathsf{head}(x_i)$, for all $1 \leq i \leq n$.

Theorem 9 (simple systems of differential equations). *A stream $\sigma \in A^\omega$ is automatic iff it is the solution of a simple zero-consistent system of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations.*

Proof. The proof essentially consists of the observation that simple zero-consistent systems of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations are in one-to-one correspondence to finite zero-consistent 2-automata. □

We refer the reader to [KR10] for a description of some more general well-defined classes of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations.

Next we illustrate that one can use $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations also to define *functions* on streams. For instance, we can define the function

$$\mathsf{inv} : 2^\omega \to 2^\omega$$

which replaces 0's by 1's and 1's by 0's, by the following equations:

$$\mathsf{inv}(\sigma)(0) = 1 - \sigma(0) \qquad \mathsf{inv}(\sigma)_0 = \mathsf{inv}(\sigma_0) \qquad \mathsf{inv}(\sigma)_1 = \mathsf{inv}(\sigma_1) \tag{4}$$

where we now write $-(0)$ for $\mathsf{head}(-)$ and where we use subscripts 0 and 1 to denote the operations of even and odd. The example is simple enough to see at once that there exists precisely one function satisfying these equations. Formally, a proof can be based as before on the finality of streams: we define an (infinite) 2-automaton $(T, \langle o, n\rangle)$ by

$$T = 2^\omega$$

with observations and transitions

$$o(\sigma) = 1 - \sigma(0) \quad n(\sigma)(0) = \sigma_0 \quad n(\sigma)(1) = \sigma_1$$

We note that this automaton is zero-consistent: for all $\sigma \in 2^*$,

$$(n(\sigma)(0))(0) = (\sigma_0)(0) = (\mathsf{even}(\sigma))(0) = \sigma(0)$$

Hence there exists, by finality, a unique homomorphism $h : T \to 2^\omega$, which we use to define the function $\mathsf{inv} = h$. It is easy to check that this function is a solution of the differential equation that we started with, and that it is the only solution.

Although the defining automaton T for inv is infinite, there still is a connection with automaticity. Namely, if σ is an automatic stream then so is $\mathsf{inv}(\sigma)$. Using Theorem 8, a proof is easy: if σ is automatic, it has only finitely many (0- and 1-) derivatives. This implies that (both c_σ and) $\mathsf{inv}(\sigma)$ has only finitely many derivatives. Which implies, again by Theorem 8, that $\mathsf{inv}(\sigma)$ is automatic.

Here is an example where the right-hand sides of the equations are no longer simple but may involve the use of stream functions. Recalling the (trivial) definitions of the streams $\mathsf{zeros} = (0, 0, 0, \ldots)$ and $\mathsf{ones} = (1, 1, 1, \ldots)$ from the introduction:

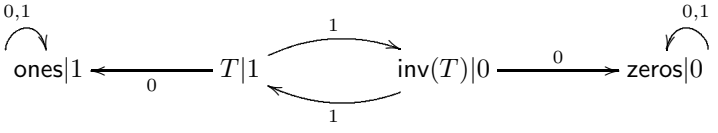
$$\mathsf{zeros}(0) = 0 \qquad \mathsf{zeros}_0 = \mathsf{zeros} \qquad \mathsf{zeros}_1 = \mathsf{zeros}$$

$$\mathsf{ones}(0) = 1 \qquad \mathsf{ones}_0 = \mathsf{ones} \qquad \mathsf{ones}_1 = \mathsf{ones}$$

and using the function inv introduced above, we consider the following differential equation:

$$T(0) = 1 \qquad T_0 = \mathsf{ones} \qquad T_1 = \mathsf{inv}(T)$$

which defines the so-called *Toeplitz* sequence. The equation can be shown to have a unique solution in a similar fashion to the examples above, by deriving from the defining differential equations for T, ones and inv a 2-automaton of the following shape:



(Here we have used the facts that $\mathsf{inv}(\mathsf{ones}) = \mathsf{zeros}$ and that $\mathsf{inv}(\mathsf{inv}(\sigma)) = \sigma$, for all σ. These facts are elementary and can be formally proved by the coinduction proof technique of Section 7.) We note hat T is automatic, since the automaton above is finite.

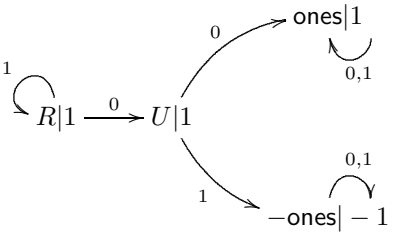
Another famous example of an automatic sequence (over the set $A = \{1, -1\}$) is the Regular Paperfolding Sequence R, which we define here by the following differential equation:

$$R(0) = 1 \qquad R_0 = U \qquad R_1 = R$$

Here U is a stream defined by

$$U(0) = 1 \qquad U_0 = \mathsf{ones} \qquad U_1 = -\mathsf{ones}$$

and minus is defined as expected, ie. $-\sigma = (-\sigma(0), -\sigma(1), -\sigma(2), \ldots)$. Again the existence of a unique solution of the equations for R and U above can be proved by finality, by the construction of an automaton of the shape



This is a zero-consistent finite 2-automaton, so both R and U are automatic.

Here is yet another example of a coinductive definition of a *function* on streams. Assuming that A has a binary operation $\cdot$ of multiplication, we define the so-called *Hadamard* product $\sigma \odot \tau$, for all streams $\sigma, \tau \in A^\omega$, by

$$(\sigma \odot \tau)(0) = \sigma(0) \cdot \tau(0) \qquad (\sigma \odot \tau)_0 = \sigma_0 \odot \tau_0 \qquad (\sigma \odot \tau)_1 = \sigma_1 \odot \tau_1$$

The product stream $\sigma \odot \tau$ consists of the elementwise products in A of the elements of σ and τ. We note that the Hadamard product preserves automaticity: if both σ and τ have only finitely many derivatives then so does $\sigma \odot \tau$.

7 Coinduction (Proofs)

A *bisimulation* between two given zero-consistent 2-automata $(S, \langle o_S, t_S \rangle)$ and $(T, \langle o_T, n_T \rangle)$ is a relation $R \subseteq S \times T$ such that for all $(s,t) \in S \times T$

$$(s,t) \in R \Rightarrow \begin{cases} o_S(s) = o_T(t) & \text{and} \\ (s_i, t_i) \in R & i = 1,2 \end{cases}$$

(where, as before, $s_i = n_S(s)(i)$ and $t_i = n_T(t)(i)$).

Equivalently, $R \subseteq S \times T$ is a bisimulation if it can be turned into a 2-automaton $(R, \langle o, n \rangle)$ such that the projections $l : R \to S$ and $r : R \to T$ (which are defined by $l((s,t)) = s$ and $r((s,t)) = t$) are homomorphisms:

$$\begin{array}{ccccc} S & \xleftarrow{\ l\ } & R & \xrightarrow{\ r\ } & T \\ {\scriptstyle \langle o_S, n_S \rangle}\downarrow & & \downarrow{\scriptstyle \langle o, n \rangle} & & \downarrow{\scriptstyle \langle o_T, n_T \rangle} \\ A \times S^2 & \xleftarrow[1_A \times l^2]{} & A \times R^2 & \xrightarrow[1_A \times r^2]{} & A \times T^2 \end{array}$$

There is the following powerful proof principle for streams.

Theorem 10 (Coinduction Proof Principle). *For any bisimulation relation $R \subseteq A^\omega \times A^\omega$ on the automaton $(A^\omega, \langle \mathsf{head}, (\mathsf{even}, \mathsf{odd}) \rangle)$ of streams, we have, for all $\sigma, \tau \in A^\omega$,*

$$\text{if } (\sigma, \tau) \in R \text{ then } \sigma = \tau$$

Proof. By the finality of A^ω and the definition of bisimulation, the projections $l, r : R \to A^\omega$ are equal. As a consequence, $\sigma = \tau$ for all $(\sigma, \tau) \in R$. □

Thus in order to show the equality of two streams, it suffices to construct a bisimulation relation that contains the pair of those streams. For a first example, we consider the function $\mathsf{zip} : A^\omega \times A^\omega \to A^\omega$ defined, for all $\sigma, \tau \in A^\omega$, by the following stream differential equation:

$$\mathsf{zip}(\sigma, \tau)(0) = \sigma(0) \quad \mathsf{zip}(\sigma, \tau)_0 = \sigma \quad \mathsf{zip}(\sigma, \tau)_1 = \tau$$

As the name suggests, this function satisfies

$$\mathsf{zip}(\sigma, \tau) = (\sigma(0), \tau(0), \sigma(1), \tau(1), \ldots)$$

We have the following trivial identity, for all $\sigma \in A^\omega$:

$$\mathsf{zip}(\sigma_0, \sigma_1) = \sigma \tag{5}$$

It can be viewed as a kind of *fundamental theorem* of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream calculus, in that it enables one to compute σ from its derivatives.

Equality (5) follows by coinduction Theorem 10 from the fact that the following relation $R \subseteq A^\omega \times A^\omega$ is a bisimulation:

$$R = \{(\mathsf{zip}(\sigma_0, \sigma_1),\, \sigma) \mid \sigma \in A^\omega\} \cup \{(\sigma,\, \sigma) \mid \sigma \in A^\omega\}$$

For another example of coinduction, we prove the following equality:

$$M = \mathsf{zip}(M, \mathsf{inv}(M))$$

where M is the Thue-Morse sequence defined in equation (3) and where inv is defined in equation (4). For a proof by coinduction, we define a relation $U \subseteq 2^\omega \times 2^\omega$ consisting of the following four pairs:

$$U = \{\, (M, \mathsf{zip}(M, \mathsf{inv}(M))),\ (M, M),\ (M, \mathsf{inv}(N)),\ (N, \mathsf{inv}(M)) \,\}$$

It is easy to check that U is a bisimulation. Thus $M = \mathsf{zip}(M, \mathsf{inv}(M))$ follows by coinduction Theorem 10.

8 Discussion

Conclusion: We have given a new perspective on automatic sequences, by defining them with a universal construction from coalgebra: finality. We have shown that our new definition is equivalent to (one of) the classical definition(s), thereby illustrating that the latter is in a precise sense universal. (Although we have talked about 2-automatic sequences only, everything can be easily adapted for k-automatic sequences, for arbitrary k.) Using our new definition, we have shown that a stream is automatic if and only if it has only finitely many (even and odd)

derivatives (Theorem 8). This characterisation offers an equivalent but convenient alternative to the more classical use of 2-kernels, as was illustrated by the examples in Section 5.

We also saw that a stream σ has only finitely many derivatives if and only if it generates a finite subautomaton $[\sigma] \subseteq A^\omega$. As a consequence, we can take the size of this subautomaton $[\sigma]$ as a well-defined notion of minimal representation for automatic sequences.

Further we have shown that a stream is automatic if and only if it can be defined by means of a simple zero-consistent system of $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations (Theorem 9). As we have seen, such equations can be used also to define functions on streams, allowing one to give easy proofs that certain functions preserve automaticity.

Related work: Our knowledge about automatic sequences is based on the aforementioned [Fog02] and [AS03], which offer a rich body of results on automatic sequences and their applications. It goes without saying that in the present paper, we have dealt with only a very small part thereof. We have explained above what might be seen as our contribution to the field of automatic sequences: a new, universal perspective on the definition of automatic sequence and, related to that, a characterisation of automaticity in terms of (even and odd) derivatives.

All the technical results of our paper are already contained in our earlier [KR10], or have been derived from there using some elementary universal coalgebra [Rut00]. What is really new is the observation that being generated by a finite 2-automaton (Definition 6) is equivalent to (the classical definition of) being 2-automatic. This observation is in itself trivial but making it gave us a kind of aha-erlebnis. And, once made, it opened the way of applying our earlier results.

Apparently, some of these insights were already in the air. After a recent presentation of our work at the COIN (Coalgebra in the Netherlands) seminar, we were told of ongoing research by Endrullis, Grabmayer, Hendriks, Klop and Moss [EGH+11], in which some related facts have been independently established (though not in a coalgebraic framework).

Further research: We see several perspectives for further research. It would be interesting to investigate more liberal formats of stream differential equations than the *simple* systems we have considered ([KR10] contains already some first attempts). This could lead to potentially interesting extensions of the set of automatic sequences, similar to the way in which context-free languages extend regular languages.

Given the parallel between regular languages and automatic sequences, as sketched in Section 4, we intend to investigate possible definitions of what could be called languages of *automatic expressions*, and a corresponding theorem in the style of Kleene. Clearly the use of the operation of zip offers a way to translate $\{\mathsf{head}, \mathsf{even}, \mathsf{odd}\}$-stream differential equations into closed expressions, but we would also be interested in operations such as (a variant on) the Kleene star or maybe some fixed point operator.

Finally, given the well-established correspondence between coalgebra and (modal) logic, it might be worthwhile to design new logics for reasoning about automatic sequences and their properties.

References

[AS03] Allouche, J.-P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press (2003)

[Con71] Conway, J.H.: Regular algebra and finite machines. Chapman and Hall (1971)

[EGH+11] Endrullis, J., Grabmayer, C., Hendriks, D., Klop, J.W., Moss, L.: Zip-specifications and automatic sequences (2011) (forthcoming), http://arxiv.org/abs/1201.3251

[Fog02] Pytheas Fogg, N.: Substitutions in dynamics, arithmetics and combinatorics. Lecture Notes in Math., vol. 1794. Springer, Berlin (2002); edited by Berthé, V., Ferenczi, S., Mauduit, C., Siegel, A.

[KR10] Kupke, C., Rutten, J.J.M.M.: Complete sets of cooperations. Information and Computation 208(12), 1398–1420 (2010)

[Ros00] Rosu, G.: Hidden Logic. PhD thesis, University of California at San Diego (2000)

[Rut98] Rutten, J.J.M.M.: Automata and Coinduction (an Exercise in Coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)

[Rut00] Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000); Fundamental Study

[Rut05] Rutten, J.J.M.M.: A coinductive calculus of streams. Mathematical Structures in Computer Science 15, 93–147 (2005)

On Topological Completeness of Regular Tree Languages

Henryk Michalewski[1,2,⋆] and Damian Niwiński[2,⋆]

[1] Institute of Mathematics, Polish Academy of Sciences
[2] University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics
{H.Michalewski,D.Niwinski}@mimuw.edu.pl

Abstract. We identify the class of $\mathbf{\Sigma}^1_1$–inductive sets studied by Moschovakis as a set theoretical generalization of the class $(1,3)$ of the Rabin-Mostowski index hierarchy of alternating automata on infinite trees. That is, we show that every tree language recognized by an alternating automaton of index $(1,3)$ is $\mathbf{\Sigma}^1_1$–inductive, and exhibit an automaton whose language is complete in this class w.r.t. continuous reductions.

Classification. F.1.1 Models of Computation,F.4.1 Mathematical Logic, F.4.3 Formal Languages.

Keywords: Alternating automata on infinite trees, Index hierarchy, Separation property.

1 Introduction

A common feature of computational complexity theory, recursion theory, automata theory, or descriptive set theory, is that they organize their realms into various hierarchies according to their sense of complexity. The complexity levels are usually understood through some concrete examples, genuine to a complexity level. For instance, the complexity class *NL* is understood through the problem of maze, and the topological class $\mathbf{\Pi}^1_1$ through the set of well-founded (infinite) trees. Of a special interest are examples which separate complexity levels, or are conjectured to do so. This often involves some concept of *completeness*, which is also a common feature of the above theories, although the actual reductions vary from polynomial-time (or log-space) reductions in complexity theory to continuous reductions in descriptive set theory.

The introduction of the μ-calculus by Kozen [13] (anticipated by the work of Emerson and Clarke, Pratt, Park, and others, see, e.g., [3] for references) gave rise to investigation of the hierarchy induced by the alternation of the least (μ) and greatest (ν) fixed point operators. Bradfield [5] proved that this hierarchy is strict, giving also [6] a natural family of examples based on parity games [8]. This model-theoretic result yielded the strictness of another hierarchy, classifying sets

⋆ Supported by the Polish Ministry of Science grant nr N N206 567840.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 165–179, 2012.

of (infinite) trees recognizable by finite automata, first considered by Rabin [22]. More specifically, Bradfield proved the strictness of the hierarchy induced by the Rabin-Mostowski index of *alternating* automata [1], corresponding level by level to the hierarchy of the μ-calculus. The witness family consists of the so-called game tree languages $W_{\iota,k}$, $\iota \in \{0,1\}$, $\iota \leq k$ (the author [6] credits I.Walukiewicz for this example). Recall that parity games are played by two players on graphs with ranked nodes and the winner is determined by the parity of the highest rank visited infinitely often. The elements of $W_{\iota,k}$ can be, roughly, viewed as unfolding of those arenas where the *even* player has winning strategy (see Section 2 below).

Unfortunately, although the questions about finite automata are usually decidable, no method is known up to date to decide the exact level of a tree language in the hierarchy, if an automaton is given[2]. This challenge seems to be related to the search for a suitable concept of reduction (and completeness) for tree automata.

On the other hand, infinite trees can be naturally viewed as elements of a Cantor (topological) space, where the concept of continuous reduction is available and several hierarchies are well understood. In order to take advantage of this page of mathematics, we need first to accurately place tree automata into the realm of descriptive set theory.

Finite-state recognizable sets of infinite words were classified already by Landweber [14, Corollary 3.6] as Boolean combinations of F_σ sets (see also [27]). Finite automata on trees are more interesting from this perspective. They recognize some Borel sets on any finite level [24], as well as some non-Borel sets [20], although by definition cannot go beyond $\mathbf{\Delta}^1_2$. It was observed by Arnold [1] that the game tree languages $W_{\iota,k}$ are complete on the subsequent levels of the alternating hierarchy w.r.t. the Wadge (i.e., continuous) reductions; on the other hand they form themselves a Wadge hierarchy [4].

The low classes of the index hierarchy are comparable to the analytic ($\mathbf{\Sigma}^1_1$) and co-analytic classes in projective hierarchies. Rabin [23] proved that (in our current terminology) the level $(1,2)$ of the index hierarchy, corresponding to the $\nu\mu$ level in the μ-calculus, is definable in the existential fragment of $S2S$, and consequently is included in the class $\mathbf{\Sigma}^1_1$. By symmetry, the level $(0,1)$ is included in $\mathbf{\Pi}^1_1$. On the other hand, there are recognizable sets of trees of levels $(0,1)$ and $(1,2)$ complete in the classes $\mathbf{\Pi}^1_1$ and $\mathbf{\Sigma}^1_1$, respectively, w.r.t. the continuous reductions [20]; in particular $W_{0,1}$ and $W_{1,2}$ have this property. It is natural to ask if the subsequent levels of the hierarchy do also enjoy some meaningful topological extensions.

In this paper, we show that the class of $\mathbf{\Sigma}^1_1$–inductive sets forms such an extension for the level $(1,3)$ of the index hierarchy, corresponding to the level $\mu\nu\mu$ in the μ-calculus. The $\mathbf{\Sigma}^1_1$–inductive sets are those that can be obtained as the least fixed points of the monotone $\mathbf{\Sigma}^1_1$–definable operators. The concept was analyzed by Moschovakis [15,16], in the frame of a more general question how the complexity of a fixed point $\mu X.F(X)$ depends on that of an operator F.

[1] For non-deterministic automata, the result was proved earlier [18].

[2] An algorithm is known only in the case if a given automaton is deterministic [19].

This question can be traced back to Kleene's observation that the least fixed points of monotone Σ^0_1-operators remain Σ^0_1, whereas the least fixed points of monotone Π^0_1-operators subsume the class Π^1_1 (see Chapter III of [10] and references there).

We verify that the game tree language $W_{1,3}$ is $\mathbf{\Sigma}^1_1$–inductive, and then show that it is actually complete among all $\mathbf{\Sigma}^1_1$–inductive sets, by a reduction from the set of *quasi bounded* trees invented by Saint Raymond [25]. By the completeness of $W_{1,3}$ on the level $(1,3)$ [1], it implies that every tree language recognized by an alternating automaton of index $(1,3)$ is $\mathbf{\Sigma}^1_1$–inductive. We terminate by providing a game characterization of the class of $\mathbf{\Sigma}^1_1$–inductive sets, in which we explore the aforementioned completeness of $W_{1,3}$.

A similar characterization of all the levels of the μ-calculus hierarchy in terms of game quantifiers has been established by Bradfield [7] for fixed-point definable sets of natural numbers. It possibly can be adapted to sets of trees, if an appropriate extension of the concept of $\mathbf{\Sigma}^1_1$–inductive sets is made.

2 Index Hierarchy

Notation. Throughout the paper, ω stands for the set of natural numbers, which we identify with its ordinal type. We also identify a natural number $n < \omega$ with the set $\{0, 1, \ldots, n-1\}$.

The concept of alternating automaton (see [27]) is best presentable via games. A *parity game* is a perfect information game of possibly infinite duration played by two players, say Eve and Adam. We present it as a tuple $(V_\exists, V_\forall, \mathit{Move}, p_0, \mathit{rank})$, where $V_\exists$ and $V_\forall$ are (disjoint) sets of positions of Eve and Adam, respectively, $\mathit{Move} \subseteq V \times V$ is the relation of possible moves, with $V = V_\exists \cup V_\forall$, $p_0 \in V$ is a designated initial position, and $\mathit{rank} : V \to \omega$ is the ranking function which admits only a finite number of values.

The players start a play in the position p_0 and then move the token according to relation *Move* (always to a successor of the current position), thus forming a path in the directed graph (V, Move). The move is selected by Eve or Adam, depending on who is the owner of the current position. If a player cannot move, she/he looses. Otherwise, the result of the play is an infinite path in the graph, $v_0, v_1, v_2, \ldots$ Eve wins the play if $\limsup_{n\to\infty} \mathit{rank}(v_n)$, is even, otherwise Adam wins. It is known that parity games are *positionally determined*: one of the players has a winning strategy which moreover can be made *positional*, i.e., represented by a (partial) function $\sigma : V \to V$ [8,17]. We say that Eve *wins* the game if she has a winning strategy, the similar for Adam.

A full binary *tree* over a finite alphabet Σ is a mapping $t : 2^* \to \Sigma$. (Recall that $2 = \{0, 1\}$.) An *alternating parity tree automaton* running on such trees can be presented as

$$\mathcal{A} = \langle \Sigma, Q_\exists, Q_\forall, q_0, \delta, \mathrm{rank} \rangle$$

where the set of states Q is partitioned into existential states $Q_\exists$ and universal states $Q_\forall$, $\delta \subseteq Q \times \Sigma \times \{0, 1, \varepsilon\} \times Q$ is a transition relation, and $\mathit{rank} : Q \to \omega$ a *rank* function. An input tree t is accepted by $\mathcal{A}$ iff Eve has a winning

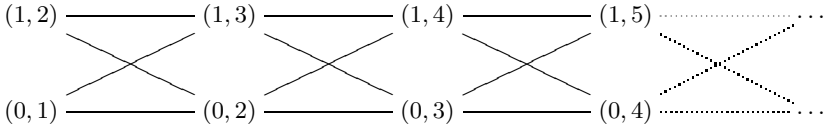


Fig. 1. The Mostowski–Rabin index hierarchy

strategy in the parity game $\langle Q_\exists \times 2^*, Q_\forall \times 2^*, (q_0, \varepsilon), \mathrm{Mov}, \Omega\rangle$, where $\mathrm{Mov} = \{((p, v), (q, vd)) \colon v \in \mathrm{dom}(t),\ (p, t(v), d, q) \in \delta\}$ and $\Omega(q, v) = \mathrm{rank}(q)$.

We assume without loss of generality that $\min \mathit{rank}(Q)$ is 0 or 1. The *Mostowski-Rabin index* is the pair $(\min \mathit{rank}(Q), \max \mathit{rank}(Q))$. It is useful to have a partial ordering on indices; it is represented on Figure 1. The idea is that we let $(\iota, \kappa) \sqsubseteq (\iota', \kappa')$ if either $\{\iota, \ldots, \kappa\} \subseteq \{\iota', \ldots, \kappa'\}$, or $\iota = 0$, $\iota' = 1$, and $\{\iota + 2, \ldots, \kappa + 2\} \subseteq \{\iota', \ldots, \kappa'\}$. We consider the indices $(1, \kappa)$ and $(0, \kappa - 1)$ as *dual*, and let $\overline{(\iota, \kappa)}$ denote the index dual to (ι, κ).

We recall an example of a witness family used by Bradfield [6] to show that the hierarchy induced by the indices of alternating parity tree automata is strict. The family consists of languages $W_{\iota,k}$, $\iota \in \{0, 1\}$, $\iota \leq k$ which are themselves based on parity games. The alphabet of $W_{\iota,k}$ is $\{\exists, \forall\} \times \{\iota, \iota + 1, \ldots, k\}$. We let $T_{\iota,k}$ denote the set of all binary trees over this alphabet. With each tree t in $T_{\iota,k}$, we associate a parity game $G(t)$, with

- $V_\exists = \{v \in 2^* : t(v) \downarrow_1 = \exists\}$,
- $V_\forall = \{v \in 2^* : t(v) \downarrow_1 = \forall\}$,
- $\mathit{Move} = \{(w, wi) \in 2^* \times 2^* : w \in 2^*,\ i \in \{0, 1\}\}$,
- $p_0 = \varepsilon$ (the root of the tree),
- $\mathit{rank}(v) = t(v) \downarrow_2$, for $v \in 2^*$.

(In the above, $\alpha \downarrow_i$, $i = 1, 2$, means the projection on the ith component.) The set $W_{\iota,k}$ consists of those trees for which Eve wins the game $G(t)$.

3 Basic Topological Concepts

All topological spaces under consideration are completely metrizable and separable. Let $\mathcal{T}_\Sigma$ denote the set of all k–ary trees over a finite alphabet Σ. This set can be equipped with a metric

$$d(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 2^{-n} \text{ with } n = \min\{|w| : t_1(w) \neq t_2(w)\} & \text{otherwise.} \end{cases}$$

It is well known and easy to see that the topological space induced by this metric is homeomorphic to the Cantor discontinuum $\{0, 1\}^\omega$. We call a *Cantor space* any space homeomorphic with $\{0, 1\}^\omega$.

The space ω^ω consists of all sequences of natural numbers. The distance between two sequences u, v is defined

$$d(u,v) = \begin{cases} 0 & \text{if } u = v \\ 2^{-n} \text{ with } n = \min\{m : u(n) \neq w(n)\} & \text{otherwise.} \end{cases}$$

For a topological space $\mathcal{H}$, the family of F_σ sets consists of all countable unions of closed sets in $\mathcal{H}$. *Borel sets* over $\mathcal{H}$ constitute the least family containing open sets and closed under complement and countable union. The *Borel relations* are defined similarly, starting with open relations (i.e., open subsets of $\mathcal{H}^n$, for some n, considered with product topology). The *analytic* (or $\mathbf{\Sigma}^1_1$) sets are those representable by

$$L = \{t \in \mathcal{H} : (\exists t')\, R(t,t')\}$$

where $R \subseteq \mathcal{H} \times \mathcal{H}$ is a Borel relation. The *co-analytic* (or $\mathbf{\Pi}^1_1$) sets are the complements of analytic sets. A continuous mapping $f : \mathcal{H} \to \mathcal{H}$ *reduces* a set $A \subseteq \mathcal{H}$ to $B \subseteq \mathcal{H}$ if $f^{-1}(B) = A$. As in complexity theory, a set $L \in \mathcal{K}$ is *complete* in class $\mathcal{K}$ if all sets in this class reduce to it.

4 $\mathbf{\Sigma}^1_1$–Inductiveness — Classical Definition

The original definition of $\mathbf{\Sigma}^1_1$–inductiveness (see [16]) refers to the least fixed points of the $\mathbf{\Sigma}^1_1$-definable operators and however formally identical to the Definition formulated below, instead of using the constructiveness in the sense of this paper, that is via arbitrary countable unions, intersections, projections and game–quantifiers, the book [16] assumes, that all the above mentioned notions are limited by the assumption of recursiveness. In Set Theory this recursive approach is called *Effective Descriptive Set Theory*, and leads to so called *lightface* classes of sets, but since recursiveness does not seem to offer at the moment any benefits to Automata Theory, we use larger and more robust classes from *Classical Descriptive Set Theory*, so called *boldface* classes. The theory of lightface and boldface classes essentially coincide when it comes to really important results, but in this article, in order to avoid any confusion, we will introduce all the boldface concepts from scratch and will not use any lightface results from [16]. In our work, we have been much inspired by [25], and our approach to $\mathbf{\Sigma}^1_1$–inductive sets follows the exposition in [25]. Also, in Section 5 we will use an interesting combinatorial example from [25] of a set complete in the class of $\mathbf{\Sigma}^1_1$–inductive.

In the definitions below C is a Cantor space and the set I is an arbitrary countable set of indices with one special element $i_0 \in I$. We will assume that I is equipped with a discrete topology. We will be mostly interested in $I = 2^\star$, $i_0 = \varepsilon$ and $C = \mathcal{T}_\Sigma$, but for certain applications in the following sections we will need the more general approach. Let

$$F : \wp(I) \times C \to \wp(I)$$

be a mapping monotone on the first argument w.r.t. the inclusion ordering. Keeping our main example in mind, if t is a tree and $I = 2^\star$, we view $F(.,t)$ as a mapping on the sets of nodes of t. We define the sets $F^\xi(t)$ by induction on ordinal ξ.

$$\begin{aligned} F^0(t) &= \emptyset \\ F^{\xi+1}(t) &= F\left(F^\xi(t), t\right) \\ F^\lambda(t) &= \bigcup_{\xi<\lambda} F^\xi(t), \text{ for limit } \lambda. \end{aligned}$$

Since F is monotone and I is countable, there is a countable ordinal ζ, such that $F^{\zeta+1}(t) = F^\zeta(t)$, and consequently $F^\zeta(t) = F^\xi(t)$, for all $\xi > \zeta$. We denote this set by $F^\infty(t)$. Finally, we let

$$\mathrm{Ind}(F) = \{t \in C : i_0 \in F^\infty(t)\} \tag{1}$$

and for $t \in \mathrm{Ind}(F)$ we define $\mathrm{rk}(t)$ as the minimal ordinal ζ such that $F^{\zeta+1}(t) = F^\zeta(t)$ and for $t \notin \mathrm{Ind}(F)$ we define $\mathrm{rk}(t) = \omega_1$.

The complexity of a mapping F is defined in terms of the relation $w \in F(Y,t)$. More specifically, we represent a set Y by its characteristic function $\chi_Y : I \to \{0,1\}$, which in turn can be viewed as an element of a Cantor space $\{0,1\}^I$.

Definition 1. A set of trees $A \subseteq C$ is $\mathbf{\Sigma}^1_1$*–inductive* if it can be presented as $A = \mathrm{Ind}(F)$, for some mapping $F : \wp(I) \times C \to \wp(I)$ (monotone on the first argument), such that the relation

$$\{(w, \chi_Y, t) \in I \times \wp(I) \times C : w \in F(Y,t)\}$$

is $\mathbf{\Sigma}^1_1$.

To show that the set $W_{1,3}$ is $\mathbf{\Sigma}^1_1$–inductive in the above sense, we have to present it as $\mathrm{Ind}(F)$, for a suitable operator F, where $C = T_{1,3}$. For a tree $t \in T_{1,3}$, a set of nodes $Y \subseteq 2^*$, and a node $w \in 2^*$, we consider a game $G(t,Y,w)$ similar to the game $G(t)$ defined on page 168 but with the following modifications. The initial position is w (rather than ε). Whenever the token arrives in a node in the set Y, the play stops. Eve wins a play $\pi = (v_0, v_1, v_2, \ldots)$ (with $v_0 = w$) if

- the label 3 can occur only at the initial position, i.e., $t(v_0) \downarrow_2 \in \{1,2,3\}$, but $t(v_i) \downarrow_2 \in \{1,2\}$, for $i \geq 1$,
- either π is finite and ends in Y,
- or π is infinite and $\limsup_{n\to\infty} t(v_n) \downarrow_2 = 2$.

We let

$$F(Y,t) = \{w \in I : \text{ Eve has a winning strategy in the game } G(t,Y,w)\}$$

Since the winning condition in $G(t,Y,w)$ is similar as in $W_{1,2}$ (i.e., of Büchi type), it is straightforward to verify that F satisfies the requirements of Definition 1. And it is not very difficult to verify that

$$\mathrm{Ind}(F) = W_{1,3}.$$

We will need later the following standard

Lemma 1. *If D, C are Cantor spaces, $\phi : D \to C$ is a continuous mapping and $A \subseteq C$ is a $\mathbf{\Sigma}^1_1$–inductive set, then the preimage $B = \phi^{-1}[A]$ is also a $\mathbf{\Sigma}^1_1$–inductive set.*

Proof. Let

$$F : \wp(I) \times C \to \wp(I)$$

be such that $A = \mathrm{Ind}(F)$, where

$$R = \{(w, \chi_Y, t) \in I \times \wp(I) \times C : w \in F(Y, t)\}$$

analytic. Define

$$G : \wp(I) \times C \to \wp(I)$$

by the formula $G(Y, t) = F(Y, \phi(t))$. Then clearly $B = \mathrm{Ind}(G)$. We have to verify that G is $\mathbf{\Sigma}^1_1$–definable.

Define

$$\psi : I \times \wp(I) \times D \to I \times \wp(I) \times C$$

by the formula

$$\psi(w, \chi_Y, t) = (w, \chi_Y, \phi(t)).$$

Then

$$\{(w, \chi_Y, t) \in I \times \wp(I) \times C : w \in G(Y, t)\} =$$
$$= \{(w, \chi_Y, t) \in I \times \wp(I) \times C : w \in F(Y, \phi(t))\} = \psi^{-1}[R].$$

Since ψ is a continuous mapping, the preimage of an analytic set R remains analytic, hence B is $\mathbf{\Sigma}^1_1$–inductive.

From the Lemma and from a result of Arnold [1] that any tree language of level (ι, k) is continuously reducible to $W_{\iota,k}$, it follows

Corollary 1. *Tree languages recognized by alternating automata of index* $(1, 3)$ *are $\mathbf{\Sigma}^1_1$–inductive.*

5 Completeness via the Quasi–Bounded Trees

We will prove that $W_{1,3}$ is complete in the class of $\mathbf{\Sigma}^1_1$–inductive sets reducing to it a set of so-called (unlabeled) *quasi-bounded trees*, whose completeness has been established by Saint Raymond [25] directly from the definition. We call a subset of ω^* an *unlabeled tree* if it closed with respect to prefixes. Characteristic function of an unlabeled tree t belongs to the Cantor space 2^{ω^*} and the topology on the space of unlabeled trees is inherited from the space 2^{ω^*}.

A tree $t \subset \omega^*$ is *cofinal* if for every $v = (v_0, v_1, \ldots) \in \omega^\omega$ there exists a branch $b = (b_0, b_1, \ldots) \in [t]$ such that $b \geq v$, that is for every $n \in \omega$ holds the inequality $b_n \geq v_n$. We define

$$QB = \{t \subset \omega^* : t \text{ is a not a cofinal tree}\}.$$

Let ψ be a mapping from 2^* to ω^* such that for a given sequence $s \in 2^*$ if $s = 0^{n_0}10^{n_1}1 \dots 10^{n_k}10^l$ $(l \geq 0)$ then

$$\psi(s) = (n_0, n_1, \dots, n_k).$$

For a given tree $t \subset \omega^*$ we define a game $\Gamma(t)$ such that Player I plays natural numbers n_0, n_1, ..., one number in every round, and Player II answers with c_0, c_1, ..., where $c_n \in \{0, 1\}$. Moreover, Player II is obliged at every step to preserve the following two conditions

1. if $\psi(c_0, \dots, c_k)$ has length $l \geq 1$, then

$$\psi(c_0, \dots, c_k) \geq (n_0, \dots, n_{l-1}).$$

2. $\psi(c_0, \dots, c_k) \in t$.

Player II wins if he managed to play infinitely many 1. In [25] one can find the following characterization

Theorem 1 (J. Saint Raymond). *A tree $t \subset \omega^{<\omega}$ is not cofinal if and only if Player I has a winning strategy in $\Gamma(t)$.*

Using a method of proof from [25, Theorem 3], we will prove the following

Theorem 2. *There exists ϕ which continuously reduces QB to $W_{1,3}$.*

Proof. For a given tree $t \subset \omega^{<\omega}$ we have to construct in a continuous way a tree $\phi(t) \in \mathrm{Tr}_{1,3}$ such that $\exists$ has a winning strategy in $\phi(t)$ if and only if t is not cofinal.

We will subsequently add new vertices to $\phi(t)$, starting from a partial tree s_0 consisting of $s_0(2, 2, \dots, 2) = (\exists, 1)$, $s_0(2, 2, \dots, 2, 1) = (\forall, 2)$. The essence of the inductive definition of $\phi(t)$ is depicted in Figure 2. We start from a tree s_0, then add it again in certain vertices marked in the Figure. There are some restrictions described below on placement of the vertices $(\forall, 3)$. Assume, that

- we already defined a partial tree $s_1 \subset \phi(t)$ and
- for v such that $s_1(v) = (\forall, 2)$, we already defined sequences $n_0^v, \dots, n_{k+1}^v$, $c_0^v, \dots, c_k^v$ such that conditions (1), (2) from the definition of $\Gamma(t)$ are fulfilled.

We will extend it to a partial tree $s_2 \subset \phi(t)$. For a given leaf $v \in s_1$ such that $s(v) = (\forall, 2)$, we add s_0 to the right, that is $s_2(v2w) = s_0(w)$. In every new leaf $v' \in s_1$ such that $s_2(v') = (\forall, 2)$, we define $n_i^{v'} = n_i^v$ $(i \leq k+1)$, $c_i^{v'} = c_i^v$ $(i \leq k)$ and $n_{k+2}^{v'}$ is the number of 2 from v to v' (formally speaking $n_{k+2}^{v'}$ is defined as $|v'| - |v| - 1$) and $c_{k+1}^{v'} = 0$.

To the left we add

1. $s_2(v1) = (\forall, 3)$, but only under the condition, that sequences $(n_0^v, \dots, n_{k+1}^v)$, $(c_0^v, \dots, c_k^v, 1)$ fulfill conditions (1), (2) from the definition of $\Gamma(t)$. We add $s_2(v12r) = s_0(r)$ and for a new leaf v' such that $s_2(v') = (\forall, 2)$ we define sequences as above with the exception, that $c_{k+1}^{v'} = 1$,

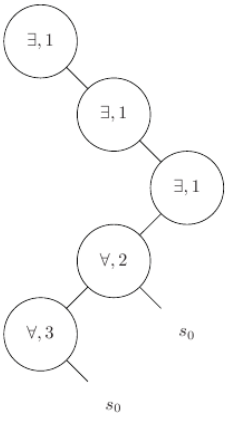


Fig. 2. An approximation of the tree $\phi(t)$

2. otherwise we add $s_2(v1r) = (\exists, 2)$ for every r; this choice ensures that Player $\forall$ will not be tempted to enter this subtree.

Claim. If Player $\exists$ has a winning strategy in $\phi(t)$, then $t \in QB$.

Proof. Assume $\exists$ has a winning strategy. Player $\exists$ plays n_0 and later in every place such that $\forall$ has a choice of going left or right, $\exists$ answers with a natural number. It means, that strategy for $\exists$ defines a mapping σ from $2^{<\omega}$ into $\omega^{<\omega}$. This extends to a continuous $\overline{\sigma}$ from 2^ω into ω^ω and bound of the image is a quasi–bound of the whole tree t.

Claim. If Player $\forall$ has a winning strategy, then t is a cofinal tree.

Proof. Take any $w \in \omega^\omega$ and play it as $\exists$. Answers of $\forall$ will lead us to a sequence in t dominating w.

This finishes the proof of the Theorem.

6 Topological Games

Now we will consider $W_{1,3}$ from a different angle, namely from the point of view of topological games and game quantifiers. We first recall the concepts of the Gale-Stewart game and the game quantifier (see, e.g., [12]). Let Y be an arbitrary set and let $A \subseteq Y^\omega$. The game $\Gamma(A)$ is played by two players, I and II, who consecutively select elements of ω.

I	y_0		y_2		y_4		y_6	$\cdots$
II		y_1		y_3		y_5		$\cdots$

The result of a play is thus an infinite sequence $y_0 y_1 y_2 \ldots$. Player I wins the play if this sequence belongs to A; otherwise II is the winner. We define $\Gamma'(A)$ as the same game with the same winning conditions, except that the first move belongs to Player II.

I		y_1		y_3		y_5	$\cdots$
II	y_0		y_2		y_4		$\cdots$

For a set C and $A \subseteq C \times Y^\omega$, we let

$$\Game(A) = \{x \in C : \text{ I has a winning strategy in the game } \Gamma(A_x)\}$$

(where $A_x = \{w : (x, w) \in A\}$).

Definition 2. A subset B of a Cantor space C is game–definable, if it can be presented as $\Game(A)$, where

$$A \subseteq C \times \omega^\omega$$

is an F_σ set in the product space of $C \times \omega^\omega$.

This concept has been analyzed by Y. N. Moschovakis [16] for the lightface sets and the equivalence of this notion and lightface notion of $\mathbf{\Sigma}^1_1$–inductiveness is the content of Exercise 7.C.10 in [16] (a conjunction of a Theorem of Wolfe and a Theorem of Solovay). The equivalence of these two notions in the boldface sense follows from [25]. We will not use directly any of this results, however certainly they inspire the following proofs and in Section 5 we already used completeness of QB in the class of $\mathbf{\Sigma}^1_1$–inductive sets proved in [25], which is part of the proof that the two notions coincide.

Proposition 1. *$W_{1,3}$ is game–definable.*

Proof. We cannot directly use the parity game from definition of $W_{1,3}$, as the parity condition of index $(1, 3)$ is not F_σ. Instead, for a tree $t \in T_{1,3}$, we consider the following modification of the game $G(t)$. The players move as previously except in the case when *rank* of the actual position is 1. In this case, Eve must exhibit a (finite) strategy to reach a node with a rank *greater* than 1 in finite time. (If it is not possible, Eve cannot win in $G(t)$.) Next, Adam chooses one of the nodes reachable by Eve's strategy, and the game continues.

To make it precise, we first define a *local strategy* (for Eve) at a node v of t. It is a *finite* subset S of the descendants of v, such that $v \in S$ and, whenever $w \in S$, then

- if $t(w) = (\forall, 1)$ then $w0, w1 \in S$;
- if $t(w) = (\exists, 1)$ then w has exactly one successor in S;
- if $t(w) \downarrow_2 \geq 2$ then w is a *leaf*, i.e., has no successor in S.

Now, for a tree $t \in T_{1,3}$, we define a parity game $H(t)$, as follows. The positions of $H(t)$ include all *tree positions* $v \in 2^*$. We distinguish between (≥ 2)-*positions*,

for which $t(v)\downarrow_2 \geq 2$, and 1-*positions*, for which $t(w)\downarrow_2 = 1$. The (≥ 2)-positions are assigned to Eve or Adam depending on the value of $t(w)\downarrow_1$; the 1-positions are always assigned to Eve. Additionally, there are *strategy positions* of the form (v, S), where v is a 1-position and S is a local strategy from v; they are assigned to Adam. The moves from the (≥ 2)-positions are the same as in $G(t)$ (to successors). From a 1-position v, there is a move to each strategy position (v, S). From a strategy position (v, S), there is a move to each tree position w, such that w is a leaf of S (note that w is then a (≥ 2)-position). The rank of a tree position with $t(v)\downarrow_2 = 3$ is 1; all other positions have rank 0. It is straightforward to see that Eve has a winning strategy in $G(t)$ iff she has a winning strategy in $H(t)$. In order to present $W_{1,3}$ in the form required in Definition 2, we have to make sure that the players move in alternation, (which needs not be the case in $G(t)$); this can be easily achieved by inserting some trivial moves. Then we define the relation A as the set of pairs (t, π), such that t is a tree in $T_{1,3}$ encoded as element of 2^ω, and π is a winning path in $H(t)$ encoded as element of ω^ω. Note that the winning condition in $H(t)$ requires that the (new) rank 1 is encountered only finitely often. Therefore, we can present A as the union of sets A_n, where A_n consists of those pairs (t, π), where π encounters 1 at most n times. The last set is closed (provided that we encode the positions of the game by natural numbers, not by sequences of bits). Hence A is F_σ, as required.

Let us notice, that as in the case of $\mathbf{\Sigma}^1_1$–inductive sets holds the following

Lemma 2. *If C, D are Cantor spaces and B is a preimage of A under a continuous mapping $\phi : D \to C$ and A is game–definable, then B is also game–definable.*

Proof. Indeed, if $A = \eth(R)$, $R \subset C \times \omega^\omega$ then $B = \eth(\{(x, y) \in C \times \omega^\omega : (\phi(x), y) \in R\})$. The second relation is itself a continuous inverse image of R, hence its complexity is not higher than that of R.

From Section 5 we know that $W_{1,3}$ is complete in the family of all $\mathbf{\Sigma}^1_1$–inductive sets and from Corollary 1 we know, that tree languages recognized by alternating automata of index $(1, 3)$ are $\mathbf{\Sigma}^1_1$–inductive. Hence we are getting the following known result (see [16, Exercise 7.C.10] and [25]):

Corollary 2. *Every $\mathbf{\Sigma}^1_1$–inductive set is game–definable. In particular, tree languages recognized by alternating automata of index $(1, 3)$ are game–definable.*

Our next goal is to show that the set $W_{1,3}$ is actually complete in the class of game–definable sets. To this end, we consider another variant of topological games, where the players select bits instead of natural numbers at the expense of a more complex winning criterion. The following concept will be useful.

Definition 3. A *parity coloring* (or coloring, for short) over a finite set Σ is a mapping $K : \Sigma^* \to \omega$, taking only a finite number of values. This coloring defines a set

$$[K] = \{u \in \Sigma^\omega : \limsup_{n\to\infty} K(u \restriction n) \text{ is even } \}$$

(where $u \restriction n = u(0) \dots u(n-1)$).

Recognition by coloring generalizes recognition by deterministic parity automata on infinite words. A related concept for finite words has been considered by Séverine Fratani [9] under the name of *automates à oracles*. A more general concept of Borel automata appears in [21]. The formulation in Definition 3 comes from the Master thesis of Michał Skrzypczak [26].

If the underlying set is a product, e.g., $\Sigma = 2 \times 2$, we identify a set $A \subseteq \Sigma^\omega$ with the relation

$$\{(x, y) \in 2^\omega \times 2^\omega : (x_0, y_0), (x_1, y_1), \ldots \in A\}.$$

Hence, a coloring $K : (2 \times 2)^* \to \omega$ induces the set $\Im([K])$ which, by definition, consists of those x, for which Player I can ensure that the result y of the play satisfies

$$\limsup_{n\to\infty} K(x \restriction n, y \restriction n) \text{ is even.}$$

Theorem 3. *The following conditions are equivalent for $A \subseteq 2^\omega$.*

1. *A is game–definable,*
2. *$A = \Im([K])$ for a coloring $K : (2 \times 2)^* \to \omega$, which takes the values in $\{1, 2, 3\}$.*

Proof. To show (2) $\Rightarrow$ (1), we prove that any set of the form $A = \Im([K])$ of (2) can be continuously reduced to $W_{1,3}$. For $x \in 2^\omega$, we define a labeled tree $t_x^K : 2^* \to \{\exists, \forall\} \times \{1, 2, 3\}$, by

$$t_x^K(v) = \begin{cases} (\exists, K(x \restriction |v|, v)) \text{ for } |v| \text{ even} \\ (\forall, K(x \restriction |v|, v)) \text{ for } |v| \text{ odd.} \end{cases} \tag{2}$$

Clearly the mapping $x \mapsto t_x^K$ is continuous and it is straightforward to see that $x \in A \iff t_x^K \in W_{1,3}$; indeed the winning strategies can be transferred easily between the two games. Since the class of game–definable sets is closed under continuous reductions and we know from Proposition 1 that it contains $W_{1,3}$, it follows that it contains A as well.

To prove (1) $\Rightarrow$ (2), suppose $A = \Im(R)$, for some $R \subseteq 2^\omega \times \omega^\omega$ of the class F_σ. We use a well-known fact that, generally, a set $E \subseteq X^\omega$ is of the class F_σ iff for some set of *finite* words $Z \subseteq X^*$ the following characterization holds (see, e.g., Theorem III 3.11 in [21]):

$$u \in E \Leftrightarrow \{n \in \omega : u \restriction n \in Z\} \text{ is finite.}$$

Let $Z \subseteq (2 \times \omega)^*$ be such a set for R. Before defining the coloring K formally, we describe the game it should induce. This game simulates the game $\Gamma(R_x)$ but, instead of natural numbers, the players choose now only bits in 2. Suppose a play in $\Gamma(R_x)$ was I: m_0, II: m_1, I: m_2, II: m_3, and so on. The choice of the number m_0 by player I is now simulated by m_0 consecutive rounds in which I chooses 0, and II answers by 0 as well. (If II violates this rule, he will be forced to loose by suitable coloring.) After this phase, I plays 1, to which II answers 0 again. Then the roles exchange: now Player I plays only 0's, to which player

II answers by 0 for m_1 times, and decides to put 1 afterward, and so on. For example, a play

$$\begin{array}{lccccc} \text{I} & 3 & & 2 & & \cdots \\ \text{II} & & 0 & & 2 & \cdots \end{array}$$

is simulated by

$$\begin{array}{lccccccccccccc} \text{I} & 0 & & 0 & & 0 & & 1 & & 0 & & 0 & & 1 & & 0 & & 0 & & 0 & & \cdots \\ \text{II} & & 0 & & 0 & & 0 & & 0 & & 1 & & 0 & & 0 & & 0 & & 0 & & 1 & \cdots \end{array}$$

A player who deviates from the above rules, looses. So a "correct" play must be of the form

$$0^{2m_0}\,10\,0^{2m_1}\,01\,0^{2m_2}\,10\,\ldots \tag{3}$$

We say that a (finite) word $0^{2m_0}\,10\,0^{2m_1}\,01\,\ldots 0^{2m_k}\,10$, *represents* the sequence $m_0m_1\ldots m_k$ if k is even; similarly, a word $0^{2m_0}\,10\,0^{2m_1}\,01\,\ldots 0^{2m_k}\,01$ represents the sequence $m_0m_1\ldots m_k$ if k is odd.

To guarantee that our coloring game simulates the game $\Gamma(R_x)$, player I should win the play (3) exactly in the case when the (pair) sequence

$$(x,\,(m_0,m_1,m_2,\ldots))$$

has only a finite number of prefixes in Z. We now define a coloring K to ensure this property. Consider $(\alpha,\beta)\in(2\times 2)^*$. We let $K(\alpha,\beta)=3$, whenever β contains a prefix which violates the rules. If β represents a sequence $m_0m_1\ldots m_k$, we let $K(\alpha,\beta)=2$ if $(\alpha\restriction(k+1), m_0m_1\ldots m_k)$ is not in Z, and $K(\alpha,\beta)=3$, otherwise. In all other cases, we let $K(\alpha,\beta)=1$ when we simulate m_k for even k and $K(\alpha,\beta)=2$ when we simulate m_k for odd k. It is then straightforward to see that $x\in\partial(R)$ iff $x\in\partial([K])$, which yields the desired presentation of A.

The equivalence of Theorem 3 along with the fact established in the proof that all sets representable as in condition (2) of Theorem 3 reduce to $W_{1,3}$ yield the following.

Corollary 3. *The set $W_{1,3}$ is complete in the class of game–definable sets.*

Finally we are getting:

Corollary 4. *The following three concepts of definability coincides for $A\subseteq 2^\omega$:*

1. *A is $\mathbf{\Sigma}^1_1$–inductive,*
2. *A is game–definable,*
3. *$A=\partial([K])$ for a coloring $K:(2\times 2)^*\to\omega$, which takes the values in $\{1,2,3\}$.*

Proof. The last two conditions are equivalent according to Theorem 3. If A is $\mathbf{\Sigma}^1_1$–inductive, then from Corollary 2 it is game–definable. If A is game–definable, then according to Corollary 3 it is reducible to $W_{1,3}$, hence from Lemma 1 the set A is $\mathbf{\Sigma}^1_1$–inductive as a preimage of $\mathbf{\Sigma}^1_1$–inductive set $W_{1,3}$.

7 Conclusion and Further Work

We have provided an evidence that the class of $\mathbf{\Sigma}^1_1$–inductive sets is a set–theoretical generalization of the class of regular tree languages of index $(1,3)$. This extends the previously known relations between $(0,1)$ vs. $\mathbf{\Pi}^1_1$ and $(1,2)$ vs. $\mathbf{\Sigma}^1_1$. Plausibly, this characterization can go further with an appropriate extension of the concept of inductiveness (in the spirit of [7]).

A related topic is the separation property, which is one of the central issues in descriptive set theory. It is known that the class of $\mathbf{\Sigma}^1_1$–inductive fails this property ([16, Theorem 6.D.4]); that is, there exist two disjoint $\mathbf{\Sigma}^1_1$–inductive sets $A, B \subseteq \{0,1\}^\omega$ such that there is no set C which would separate A and B and which would simultaneously satisfy the conditions that C and $\{0,1\}^\omega \setminus C$ are $\mathbf{\Sigma}^1_1$–inductive sets.

We established recently in a joint paper with André Arnold [2] the failure of separation property for all the levels (ι, n) of the alternating index hierarchy, for n odd (for the level $(0,1)$, the result was known [11]). We are planning to establish whether the pair of regular languages of index $(1,3)$ constructed in [2] could serve as an example of inseparable pair for the class of $\mathbf{\Sigma}^1_1$–inductive sets. The question if the separation property holds for the levels (ι, n) with n even remains open (it is only known to hold for $(1,2)$ [11,23]). One may hope that the ideas from descriptive set theory may offer an insight into the problem.

References

1. Arnold, A.: The μ-calculus alternation-depth hierarchy is strict on binary trees. RAIRO-Theoretical Informatics and Applications 33, 329–339 (1999)
2. Arnold, A., Michalewski, H., Niwiński, D.: On the separation question for tree languages. In: STACS 2012 (to appear, 2012)
3. Arnold, A., Niwiński, D.: Rudiments of μ-Calculus. Elsevier Science, Studies in Logic and the Foundations of Mathematics, vol. 146. North-Holland, Amsterdam (2001)
4. Arnold, A., Niwiński, D.: Continuous separation of game languages. Fundamenta Informaticae 81(1-3), 19–28 (2007)
5. Bradfield, J.C.: The modal μ-calculus alternation hierarchy is strict. Theoret. Comput. Sci. 195, 133–153 (1997)
6. Bradfield, J.C.: Simplifying the Modal μ-Calculus Alternation Hierarchy. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 39–49. Springer, Heidelberg (1998)
7. Bradfield, J.C.: Fixpoints, games and the difference hierarchy. RAIRO-Theoretical Informatics and Applications 37, 1–15 (2003)
8. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proceedings 32th Annual IEEE Symp. on Foundations of Comput. Sci., pp. 368–377 (1991)
9. Fratani, S.: Regular sets over tree structures. Theoret. Comput. Sci. 418, 48–70 (2012)
10. Hinman, P.G.: Recursion-Theoretic Hierarchies. Perspectives in Mathematical Logic. Springer, Heidelberg (1978)

11. Hummel, S., Michalewski, H., Niwiński, D.: On the Borel Inseparability of Game Tree Languages. In: STACS 2009, pp. 565–575 (2009)
12. Kechris, A.S.: Classical descriptive set theory. Springer, New York (1995)
13. Kozen, D.: Results on the propositional μ-calculus. Theoret. Comput. Sci. 27, 333–354 (1983)
14. Landweber, L.H.: Decision problems for ω-automata. Math. Systems Theory 3, 376–384 (1969)
15. Moschovakis, Y.N.: Elementary Induction on Abstract Structures. North Holland (1974)
16. Moschovakis, Y.N.: Descriptive Set Theory. North Holland (1980)
17. Mostowski, A.W.: Games with forbidden positions. Technical Report 78, Instytut Matematyki, University of Gdańsk (1991)
18. Niwiński, D.: On Fixed Point Clones. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 464–473. Springer, Heidelberg (1986)
19. Niwiński, D., Walukiewicz, I.: Deciding Nondeterministic Hierarchy of Deterministic Tree Automata. Electr. Notes Theor. Comput. Sci. 123, 195–208 (2005)
20. Niwiński, D., Walukiewicz, I.: A gap property of deterministic tree languages. Theoret. Comput. Sci. 303, 215–231 (2003)
21. Perrin, D., Pin, J.-E.: Infinite words. Elsevier, Amsterdam (2004)
22. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Soc. 141, 1–35 (1969)
23. Rabin, M.O.: Weakly definable relations and special automata. In: Bar-Hillel, Y. (ed.) Mathematical Logic and Foundations of Set Theory, pp. 1–23 (1970)
24. Skurczyński, J.: The Borel hierarchy is infinite in the class of regular sets of trees. Theoret. Comput. Sci. 112, 413–418 (1993)
25. Saint Raymond, J.: Quasi–bounded trees and analytic inductions. Fundamenta Mathematicae 191, 175–185 (2006)
26. Skrzypczak, M.: O kolorowaniach drzewa Cantora. Master Thesis, University of Warsaw (2010)
27. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Heidelberg (1997)

A Complete Logical System for the Equality of Recursive Terms for Sets

Lawrence S. Moss[1], Erik Wennstrom[2], and Glen T. Whitney[3]

[1] Department of Mathematics,
Indiana University, Bloomington, IN, USA 47405
lsm@cs.indiana.edu
[2] School of Informatics and Computing,
Indiana University, Bloomington, IN, USA 47405
ewennstr@indiana.edu
[3] Museum of Mathematics, 134 West 29th Street,
Suite 709/710 New York, NY, 10001
whitney@momath.org

Abstract. This paper presents a sound and complete logical system whose atomic sentences are the equalities of recursive terms involving sets. There are two interpretations of this language: one makes use of non-wellfounded sets with finite transitive closure, and the other uses pointed finite graphs modulo bisimulation. Our logical system is a sequent-style deduction system. The main axioms and inference rules come from the FLR_0-proof system from [6], including the Recursion Inference Rule (but an additional axiom is needed), and also axioms corresponding to the extensionality axiom of set theory.

1 Introduction: Fixed Point Terms for Sets

This paper presents a language of *recursive terms for sets*. We have in mind terms like the following:

$$x \texttt{ where } \{x = \{x, y\}, y = \emptyset\} \tag{1}$$

$$x \texttt{ where } \{x = \{z, y\}, y = \emptyset, z = \{x, y\}\} \tag{2}$$

$$\{x, y\} \texttt{ where } \{x = \{x, y\}, y = \emptyset\} \tag{3}$$

$$\{\, x \texttt{ where } \{x = \{x, y\}, y = \emptyset\}\,,\; y \texttt{ where } \{x = \{x, y\}, y = \emptyset\}\,\} \tag{4}$$

$$z \texttt{ where } \{x = x, y = \{x\}, z = \{y\}\} \tag{5}$$

$$x \texttt{ where } \{x = x\} \tag{6}$$

$$x \texttt{ where } \{x = \{y\}\} \tag{7}$$

We propose a formal language with these as terms; it will be a variant of the language FLR_0 from [6] where we replace arbitrary function symbols with set-forming operations. The idea is to interpret terms such as (1–7) above as sets.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 180–203, 2012.

For example, consider the term in (1) above. Assuming the Anti-Foundation Axiom (AFA) introduced by Forti and Honsell [5] (see also Aczel [1] and Barwise and Moss [3]), there is a unique set a which satisfies the condition that $a = \{a, \emptyset\}$. We therefore interpret the term in (1) as this set a.

If one works in the usual set theory *ZFC*, then due to the Foundation Axiom there is no set $a = \{a, \emptyset\}$. But even in this setting it makes sense to consider terms like (1), but then one would want to take the solution to be a graph rather than a set. It would be natural to take the solution to be the graph below:

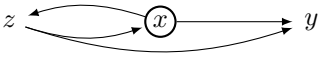
(8)

Since (1) mentions x as its head term (the one before `where`), we really need to take our solution to be a a *pointed* graph (a graph together with a specified point). The circle indicates the specified point.

To have a unique solution, one would want to identify graphs modulo isomorphism. It is even sensible to identify graphs modulo bisimulation. The reason for this is suggested by (2) and the set semantics. As a set, the solution to (2) would be the unique set a solving the system

$$\begin{aligned} x &= \{z, y\} \\ y &= \emptyset \\ z &= \{x, y\} \end{aligned} \tag{9}$$

In more detail, AFA tells us that there is a unique solution s to this system. This solution is a function of the variables x, y, and z, and the requirement on s is that $s_x = \{s_z, s_y\}$, $s_y = \emptyset$, and $s_z = \{s_x, s_y\}$. We interpret (2) by s_x. Recalling the notion of *bisimulation on sets* (see [3]), the sets $a = \{a, \emptyset\}$ interpreting (1) and s_x solving (9) are bisimilar, and hence assuming AFA, they are equal. In terms of graphs, the solution to (2) would be



(10)

There is a bisimulation relation between the underlying graph of (8) and that of (10). As a set of ordered pairs, this relation is

$$\{\langle x, x\rangle, \langle x, z\rangle, \langle y, y\rangle\}$$

This bisimulation relates the distinguished points of the two structures. All of this is what is meant by *pointed graphs modulo bisimulation.*

Please note that we use the set braces { and } in two different ways in a term like x `where` $\{x = \{x, y\}, y = \emptyset\}$; first, as set braces in $\{x, y\}$, and second as a delimiter of the two-equation system on the right of the `where` $\{\}$ construct. The reason that we need some sort of notation in connection with recursive terms is that we shall allow recursive terms inside of recursive terms, and without the extra brackets the resulting syntax would not be uniquely parsable.

Returning to our examples, the denotation of (3) should be $\{a, b\}$, where a is the denotation of x in (1), and b the denotation of y in the same term (1).

We would like a language rich enough to allow us to take *finite sets of terms*, as in (4). The natural denotation of this term should be the same as that of (3), and so also the same as that of (1) and (2).

The term in (5) presents us with something of a problem. On the one hand, the system $x = x$ does not have a *unique* solution; indeed, *every* set solves this equation. (This has nothing to do with AFA.) So when we consider our language of terms, we either must ban such a term, or else chose some arbitrary "scapegoat" for systems like this. Both of these alternatives are worth considering in a logical system, and the details will turn out to be different. Our choice in this paper is the second one; we want to take a set $\bot$ and arrange that the denotation of (5) be $\bot$. This means that the denotation of term (6) will be $\{\{\bot\}\}$.

Our last issue in crafting a syntax of recursive terms has to do with free variables, as in (7). For that matter, it is natural to take variables themselves to be terms, and so we must also give denotations to variables. It is customary to define denotations relative to assignment functions (these are functions from variables to some underlying universe). But in the set theoretic context, variables are themselves sets, and we might as well take the denotation of a variable to simply be itself. Following this, the denotation of (7) will be $\{y\}$.

We shall shortly turn to the formal semantics of terms in our language. Before that, let us mention what we want to do with those terms in this paper. First, we would like a logical system that allows us to prove equalities such as the one we saw above for (1) and (2). That is, we want to arrange that

$$\vdash\ x\ \texttt{where}\ \{x = \{x, y\}, y = \emptyset\} \quad = \quad x\ \texttt{where}\ \{x = \{z, y\}, y = \emptyset, z = \{x, y\}\}\,.$$

Our proof system is based on that of FLR_0 from [6], and a crucial feature is proofs with hypotheses. So we want to say, for example,

$$x = y \vdash\ z\ \texttt{where}\ \{z = \{x, z\}\}\ =\ z\ \texttt{where}\ \{z = \{y, z\}\}\,.$$

For that matter, we want

$$x = y \vdash \{z, x\} = \{z, y\}$$

In words, no matter what sets x, y, and z are, if $x = y$, then $\{x, z\} = \{y, z\}$. This will correspond to a soundness assertion,

$$x = y \models \{z, x\} = \{z, y\},$$

and of course we need a formal semantics of terms for this. We also wish to incorporate some very elementary set-theoretic reasoning in our logical system. This goes beyond the more general work done in [6]. For example, we want our proof system to reflect instances of Extensionality such as the following: if $\{x\} = \{y, z\}$, then either $x = y$ or $x = z$. For this, our proof system will be a *sequent calculus*: we shall arrange that

$$\{x\} = \{y, z\} \vdash x = y, x = z$$

As is customary, the comma on the right will be interpreted disjunctively; commas on the left will be conjunctive, as in

$$x = y, y = z \vdash x = z$$

The main point of the paper is to provide a sound and complete proof system for recursive terms which describe sets under AFA (equivalently, pointed graphs modulo bisimulation), and which incorporates reasoning using Extensionality.

2 The language: Syntax of Terms, Equations, and Formulas

Fix a set of *variables*

$$\mathsf{V} \quad = \quad \{x_1, x_2, \ldots, x_n, \ldots\}.$$

We always assume that $x_i \neq x_j$ for $i \neq j$. For any set S, let $(\mathsf{V} \to S)$ denote the set of all functions σ whose domain is a finite subset of V with the property that $\sigma(x) \in S$ for all $x \in dom(\sigma)$. Then the set T of *terms* of our language $\mathcal{L}$ is defined to be the least fixed point of the following (set) equation:

$$T \quad = \quad \mathsf{V} + \mathcal{P}_{fin}T + \big(T \times (\mathsf{V} \to T)\big). \tag{11}$$

Since the right-hand side of this equation is a monotone operator on T, such a least fixed point exists.

We always suppress the injections into the coproduct of (11). So we regard V as a subset of T, and we also regard each finite subset of T as an element of T. We use `where` in connection with the last component of the sum, to denote recursive terms. For example, instead of writing

$$\langle \{x, z\}, \{\langle x, \{y, z\}\rangle, \langle y, \emptyset\rangle, \langle z, \{z\}\rangle\}\rangle,$$

we write $\{x, z\}$ `where` $\{x = \{y, z\}, y = \emptyset, z = \{z\}\}$.

We have the usual notions of free and bound variables. If E is the term A `where` $\{\sigma\}$, then the variables in the domain of σ are bound everywhere in E; variables which are not bound by any occurrence of `where` are free.

A *term substitution* is a function $\sigma : \mathsf{V} \to T$. We define its extension $[\sigma] : T \to T$ by recursion:

$$\begin{array}{lcl}
[\sigma]x & = & \sigma(x) \\
[\sigma]\{A_1, \ldots, A_n\} & = & \{[\sigma]A_1, \ldots, [\sigma]A_n\} \\
[\sigma](\, A \texttt{ where } \{\boldsymbol{x} = \boldsymbol{A}\}\,) & = & ([\tau]A) \texttt{ where } \{\boldsymbol{x} = [\tau]\boldsymbol{A}\} \\
 & & \quad \text{where } \tau(y) = y \text{ for } y \in \boldsymbol{x}, \\
 & & \quad \text{and } \tau(y) = \sigma(y) \text{ for } y \in dom(\sigma) - \boldsymbol{x},
\end{array}$$

An *atomic formula* of $\mathcal{L}$ is an equation $A = B$ between terms of $\mathcal{L}$. A *formula* is an atomic formula, perhaps universally quantified by a sequence of distinct variables. So we would write $\forall \boldsymbol{x}(A = B)$ for a formula.

3 Semantics of $\mathcal{L}$

At this point, we have spelled out the syntax of terms of $\mathcal{L}$, and also the formulas of this language. The time has come to go into details on the semantics. Our discussion in Section 1 mentioned that the terms could be interpreted either as sets (assuming AFA) or as pointed finite graphs modulo bisimulation. We are going to present the semantics both ways.

The first presentation of the semantics uses finite graphs modulo bisimulation. We feel that all readers should be able to follow this most concrete semantics. Then we re-present the semantics using non-wellfounded sets, for those familiar with the topic.

Before we give the semantics, we need some points that are actually about the syntax.

Definition 3.1. *Let E be any term, say*

$$E \quad \equiv \quad E_0 \texttt{ where } \{x_1 = E_1, \ldots, x_n = E_n\}$$

The E-sequence from x_i *is the longest finite or infinite sequence of variables taken from the list x_1, …, x_n, say*

$$x_i = y_0, y_1, \ldots, y_n, \ldots$$

such that the body of E contains the equation $y_{j-1} = y_j$ for every $j > 0$. That is, for every $j > 0$, the right-hand side of the equation for the variable y_{j-1}) is the variable y_j.

We say that x_i is ungrounded in E *if the E-sequence from x_i is infinite. Otherwise, x_i is* grounded in E*. In such a case, if the E-sequence from x_i ends in x_j, then we have information about the corresponding term E_j in E: E_j is either a variable outside of $\{x_1, \ldots, x_n\}$ (that is, a free variable in the the overall term E), or E_j is a recursion term (one containing* `where`*), or else E_j is a set term.*

Example 3.2.

$$\{\emptyset, \{x, y, v\}\} \texttt{ where } \{x = z, y = \{y, w\}, z = x, w = \{y\}\} \tag{12}$$

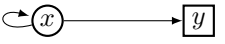
We have made life a little easier by using different variables instead of subscripts. We continue the practice and write $E_x = z$, $E_y = \{y, w\}$, $E_z = x$, and $E_w = \{y\}$.

The E-sequence from x is $x, z, x, z, \ldots$. The E-sequence from z is $z, x, z, x, \ldots$. So x and z are ungrounded in E. The E-sequences from y and w are finite and end in set terms.

3.1 Semantics in Pointed Finite Graphs Modulo Bisimulation

Our first semantics uses pointed finite graphs modulo bisimulation. To save on some terminology, in what follows, *graph* means *finite pointed graph modulo bisimulation.* Graphs for us allow self-loops but not for multiple edges between points.

Let $Y \subseteq \mathsf{V} \cup \{\bot\}$. A *graph over* Y is a graph where some of the end nodes are labeled in the set Y. We indicate end nodes in graphs over Y by putting a box around them. These must be end nodes of the graph; they cannot have children. Consequently, if the root node belongs to Y, then that node is the entire graph. For example, here is a graph over $\{y\}$:



It will turn out that this graph is

$$\| \, x \ \texttt{where} \ \{x = \{x, y\}\} \, \| \, .$$

For each term A with free variables in $Y = \{y_1, \ldots, y_k\}$, we shall arrange that $\|A\|$ be a graph over Y.

For a variable x, we take $\|x\| = \boxed{x}$.

For a set term $\{A_1, \ldots, A_n\}$ we take the disjoint union of $\|A_1\|$, ..., $\|A_n\|$. Then we add a fresh point $*$ which we take to be the overall distinguished node. We take as the children of $*$ the distinguished points of A_1, ..., A_n. We also take the quotient of the resulting accessible pointed graph by the largest bisimulation. This means that in case $\|A_i\| = \|A_j\|$, we only want one copy.

Finally, the semantics of a `where` term such as (13) below is defined in stages:

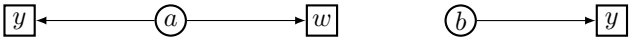
$$E \quad \equiv \quad E_0 \ \texttt{where} \ \{x_1 = E_1, \ldots, x_n = E_n\} \tag{13}$$

Write X for the finite set $\{x_1, \ldots, x_n\}$. Let Y be a finite set such which includes the free variables of E. First, for $i = 1, \ldots, n$, let G_i be the graph over $X \cup Y \cup \{\bot\}$ defined as follows. If x_i is ungrounded in E, let G_i be the one-point graph with distinguished point $\bot$. If x_i is grounded and the x_i-sequence over E leads to A, then let $G_i = \|A\|$.

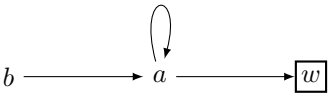
Second, define graphs H_i as follows: take as node set for H_i the disjoint union $G_1 + \cdots + G_n$. Each G_i is a pointed graph and has its own distinguished point which we'll call g_i. Identify each end node $\boxed{x_i}$ with g_i. (That is, take the quotient graph.) Declare the overall distinguished point of H_i to be g_i. Finally, take the subgraph accessible from g_i.

Finally, the semantics of E from (13) is taken to be $\|E_0\|$, with each $\boxed{x_i}$ identified with the distinguished point of G_i.

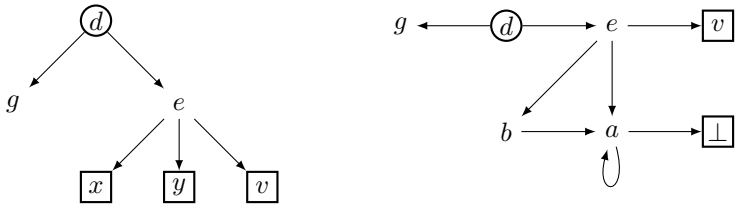
Example 3.3. We exhibit the denotation in the graph semantics for the term in (12). The set X of bound variables is $\{x, y, z, w\}$. The set Y of free variables is $\{v\}$. G_x and G_z are the same graph, the one-point graph $\boxed{\bot}$. G_y and G_w are the two pointed graphs shown below:



Here a and b are arbitrary sets. Let H be



Then H_y is this graph H, with a taken as the distinguished point (so that b is absent), and H_w is H with b as distinguished point. H_x and H_z are again $\boxed{\perp}$. Continuing, the interpretation of the head term, $\|\{\emptyset, \{x, y, v\}\}\|$, is shown on the left below:



The overall semantics of the term in (12) is obtained by substitution into the graph for $\{\emptyset, \{x, y, v\}\}$. It would be the graph on the right above.

3.2 Semantics in Non-wellfounded Sets

The most natural semantics of $\mathcal{L}$ would probably be the set semantics, since the syntax of the language uses sets rather than graphs. Readers of [3] will quickly see how the terms of $\mathcal{L}$ may be interpreted as *non-wellfounded sets*, where we use $\mathsf{V} \cup \{\perp\}$ as a set of *urelements*. (Indeed, as we mention in Remark 3.6 below, we want to add an additional countable set $\mathcal{A}$ of urelements.)

We define $\|E\|$ for terms E by recursion. Variables are interpreted by themselves: $\|x\| = x$. For set terms, we take

$$\|\{E_1, \ldots, E_n\}\| \quad = \quad \{\|E_1\|, \ldots, \|E_n\|\}.$$

For a recursion term E, we first modify E to set the ungrounded variables to $\perp$, and those variables which are grounded to a free variable outside of E to that variable itself. The body of the resulting term is then a system of equations in the bound variables. Then AFA gives a solution to this system, a function s from the bound variables. Using the work of Chapter 8 in [3], s extends to an operation $[s]$ defined on all sets, and we take $\|E\|$ to be $[s](\|E_0\|)$, where E_0 is the head term of E.

For any set or class X, $HF^{1/2}(X)$ denotes the collection of sets over X whose transitive closure is finite. An important feature of the semantics is that each term E will get a denotation in

$$Y \cup \{\perp\} \cup HF^{1/2}(Y \cup \{\perp\}),$$

where Y is the set of free variables in E.

Example 3.4. Again we consider the term from (12). We modify the term to take care of the ungrounded variables x and z, obtaining

$$\{\emptyset, \{x, y, v\}\} \texttt{ where } \{x = \perp, y = \{y, w\}, z = \perp, w = \{y\}\} \tag{14}$$

By AFA, the system on the right has a unique solution, say s. This would be a function with domain $X = \{x, y, z, w\}$ such that $s_x = \bot = s_z$, $s_y = \{s_y, s_w\}$, and $s_w = \{s_y\}$. Then s extends to $[s]$ defined on all sets built from X and $\bot$. The interpretation of the term in (12) will be

$$[s](\{\emptyset, \{x, y, v\}\}) \quad = \quad \{\emptyset, \{s_x, s_y, v\}\}.$$

Remark 3.5. For those not familar with non-wellfounded sets, here is an entry point to the discussion above. Consider the set $\mathcal{S}$ of all graphs from Section 3.1. Write $G \in H$ if there is some child h of the root of G such that H is the part of G accessible from h. (Recall that we always identify graphs up to bisimulation.) Then the resulting structure $(\mathcal{S}, \in)$ satisfies all of the axioms of set theory except two. Since our graphs are finite, $(\mathcal{S}, \in)$ will not satisfy the Axiom of Infinity. And since we are using graphs which might have cycles, the Axiom of Foundation will also not be satisfied. Indeed, $(\mathcal{S}, \in)$ satisfies a weak form of the Anti-Foundation Axiom, one which basically says that all finite systems of equations have unique solutions, where the left-hand sides are variables, and the right-hand sides are either a (finite) set of variables found on the left side, or a single variable not on the left side, or $\bot$. The identification of graphs modulo bisumulation corresponds to the Axiom of Extensionality. To summarize, the reader not familiar with non-wellfounded sets could simply work with graphs, reading $A \in B$ in the way we just described, and (as always) identifying two sets which have the same elements.

Remark 3.6. It will be convenient in several places (Lemmas 4.3 and 4.10) to work with a version of the set semantics that allows for a countable set $\mathcal{A}$ of additional urelements. So we shall work in the universe of sets built from $\mathsf{V} \cup \{\bot\} \cup \mathcal{A}$. These additional urelements are mostly needed for convenience, and we suspect that they are not strictly needed for our completeness results.

3.3 Semantics Using the Monad of Iterative Algebras

Finally, we have yet a third semantics, using the rational monad of a finitary set functor. We present this mostly to connect to the literature on coalgebra. It also gives a more general semantics, since it goes via *iterative algebras*. That is, we are using the initial iterative algebra of the finite power set functor $\mathcal{P}_{fin}$. This functor takes a set X to the set $\mathcal{P}_{fin}(X)$ of finite subsets of X. It also takes a function $f : X \to Y$ to the function $\mathcal{P}_{fin}f : \mathcal{P}_{fin}X \to \mathcal{P}_{fin}Y$ given by direct images: for a finite $X_0 \subseteq X$,

$$\mathcal{P}_{fin}(f)(X_0) \quad = \quad f[X_0] \quad = \quad \{f(x) : x \in X_0\}.$$

However, much of what we need in order to present the semantics and prove some needed results follows only from the fact that we have an iterative algebra. (Sometimes we need to know that we are working with an initial iterative algebra.) So this third presentation points the way to further results along the lines

of what we are doing in this paper, but for initial iterative algebras of functors besides $\mathcal{P}_{fin}$.

We shall write H for the finite power set functor $\mathcal{P}_{fin}$ in what follows, partly to make the notation more readable and also to pave the way for a generalization later on.

An *H-algebra* is a pair (A, a), where $a : HA \to A$. In contrast, an *H-coalgebra* is a pair (A, a), where $a : A \to HA$.

An H-algebra (A, a) is *iterative* if for every finite set X and every $f : X \to HX + A$, there is a unique $f^\dagger : X \to A$ such that the diagram below commutes:

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & HX + A \\ {\scriptstyle f^\dagger}\downarrow & & \downarrow{\scriptstyle Hf^\dagger + A} \\ A & \xleftarrow[[a,A]]{} & HA + A \end{array}$$

An early reference on iterative algebras in this sense is Nelson [8], but the paper that is closest to what we do is Adámek, Milius, and Velebil [2]. This paper proves that such algebras exist, not only for our H but for many other set functors. In fact, assuming a mild condition on set functors J (accessibility, also called boundedness), J has a *final coalgebra* $(A, a : A \to JA)$. Then a is invertible by Lambek's Lemma, and (A, a^{-1}) is not only iterative, it is *completely iterative* (it satisfies the iterativity condition even for infinite sets X).

Continuing to quote results from [2], for each set X there is a *free iterative H-algebra* RX. We have $RX = HRX + X$. (The letter R comes from the words "rational" and "regular", two sources of examples.) Moreover, R carries the structure of a monad (R, η, μ), where $\eta_X : X \to RX$ is $\mathsf{inr} : X \to HRX + X$.

For each set $X \subseteq \mathsf{V}$, and each term E all of whose free variables belong to X, we define $\|E\|_X \in RX$. The definition is by recursion on E, and the only interesting part is the one for recursion terms. Consider again the term from (13),

$$E \quad \equiv \quad E_0 \texttt{ where } \{x_1 = E_1, \ldots, x_n = E_n\}$$

Let Y be a finite set which includes the free variables of the overall term E. Let $X = \{x_1, \ldots, x_n\}$, so that all E_i have their free variables in $X \cup Y$. We already have $\|E_i\|_{X \cup Y}$ for all i. This gives a map $e : X \to R(X \cup Y) \to R(X + Y)$. By Corollary 4.7 of [2], there is a unique $e^\dagger : RY$ such that the diagram below commutes:

$$\begin{array}{ccc} X & \xrightarrow{\quad e^\dagger \quad} & RY \\ {\scriptstyle e}\downarrow & & \uparrow{\scriptstyle \mu_Y} \\ R(X + Y) & \xrightarrow[R[e^\dagger, \eta_Y]]{} & RRY \end{array}$$

Then we set

$$\|E\|_Y \quad = \quad (\mu_Y \circ R[e^\dagger, \eta_Y])\, \|E_0\|\,.$$

To obtain a more exact match between this semantics and our previous work, we would use

$$\|E\| \quad = \quad Ri(\|E\|_Y),$$

where $i : Y \to \mathsf{V}$ is the inclusion.

Remark 3.7. We have presented three semantics for the same language. To be concrete in the rest of this paper, we are going to use the second semantics, the one that uses non-wellfounded sets.

3.4 The Substitution Lemma

Each term substitution σ gives rise to another substitution $\hat{\sigma} : X \to R\mathsf{V}$ given by

$$\hat{\sigma}(x) \quad = \quad \|\sigma(x)\| .$$

Lemma 3.8. *The diagram below commutes:*

$$\begin{array}{ccc} T & \xrightarrow[\|\quad\|]{} & RY \\ {\scriptstyle [\sigma]}\downarrow & & \downarrow{\scriptstyle [\hat{\sigma}]} \\ T & \xrightarrow[\|\quad\|]{} & RY \end{array}$$

That is, for all terms A and all term substitutions σ,

$$\|[\sigma]A\| \quad = \quad [\hat{\sigma}]\,\|A\| .$$

Proof. By induction on A.

3.5 Semantics of Equations and Formulas

At this point, we have three semantics of terms, interpreting a term A by a set or graph $\|A\|$. We complete the semantics with a discussion of equations and formulas.

Let α be a substitution, a map from some subset of X to $R(\mathsf{V} + \{\bot\})$. We say that α *satisfies* the formula $\forall \boldsymbol{x}(A = B)$ if $[\beta]\ \|A\| = [\beta]\ \|B\|$, for any substitution β which agrees with α on all variables in $dom(\alpha) \setminus \boldsymbol{x}$. For example, α satisfies $\forall x(\{x, y, z\} = \{x, y\})$ exactly if $\alpha(y) = \alpha(z)$. And no substitution satisfies the equation $\forall x(x = \emptyset)$.

A closed formula is *valid* if it is satisfied by any (equivalently, all) substitutions.

More generally, we write $\Delta \models \Sigma$ to mean that for every substitution α, if α satisfies each $\phi \in \Delta$, then α satisfies some $\psi \in \Sigma$.

4 A Sequent Deduction System

We now write down a specific sequent deduction system which is sound and complete for $\Delta \models \Gamma$ for sequents (i.e., finite sets) Δ and Γ. Our proof gives the additional information that this entailment relation is decidable.

Before presenting the system, we make a few remarks. First, one important feature of the Anti-Foundation Axiom (and of iterative algebras more generally) is that the sets guaranteed as solutions to systems of equations are *unique*. So, for example,

$$f(A) = A \models A = x \texttt{ where } \{x = f(x)\}, \tag{15}$$

in the structure examined in the previous section. We must have a rule to derive the consequences of this uniqueness. It can be shown that valid semantic facts such as (15) cannot be proven in the proof system for FLR_0 from [6]; the idea behind the proof is that not every FLR_0 structure has unique fixed points for each of its transformations. It turns out that our Uniqueness Axiom will be the only new rule dealing with recursion, beyond the axiomatization of [6]. All of the other new axioms arise to handle set-theoretic phenomena, notably the Axiom of Extensionality.

The following collection of axioms and inference rules comprise our proof system for $\mathcal{L}$. For the most part, the axioms and inference rules transcribe the proof system of [6], which is complete for the standard FLR_0 identities. The only changes are as follows: The Cut rule is extended to handle multiple formulas on the right of $\vdash$; the Uniqueness axiom for recursion is new; and a few new axioms are added to handle the set-theoretic phenomena that arise.

In the following rules, A, B, C, and E are arbitrary terms, ϕ and ψ are arbitrary formulas, and Δ, Σ, Ξ, and Θ are arbitrary finite sets of formulas, unless otherwise noted. When dealing with assertions such as

$$\{\phi_1, \ldots, \phi_k\} \vdash \{\psi_1, \ldots, \psi_l\},$$

we always omit the set braces on each side and simply list the formulas.

Logical and equational axioms

- $\phi \vdash \phi$.
- $\vdash A = A$; $A = B \vdash B = A$; $A = B, B = C \vdash A = C$.
- (Replacement) $A = B \vdash [x \mapsto A](E) = [x \mapsto B](E)$, provided the substitutions are free.
- (Specialization) $\forall x(\phi(x)) \vdash \phi(E)$, provided the substitution is free.

Logical inference rules

- (Weakening) From $\Delta \vdash \Sigma$, conclude $\Delta \cup \Xi \vdash \Sigma \cup \Theta$.
- (Extended Cut) From $\Delta \vdash \Xi$ and $\Delta, \xi \vdash \Sigma$ for each $\xi \in \Xi$, conclude $\Delta \vdash \Sigma$.
- (Generalization) From $\Delta \vdash \phi(x), \Gamma$ conclude $\Delta \vdash \forall x(\phi(x)), \Gamma$, provided that x does not occur free in Δ.

Recursion axioms

- (Head) $\vdash A \texttt{ where } \{x_1 = B_1, \ldots, x_n = B_n\} = [\sigma]A$, where the term substitution σ takes x_i to $x_i \texttt{ where } \{x_1 = B_1, \ldots, x_n = B_n\}$ for each i from 1 to n.
- (Bekič-Scott) $\vdash A \texttt{ where } \{\boldsymbol{y} = \boldsymbol{C}, \boldsymbol{x} = \boldsymbol{B}\} = A' \texttt{ where } \{\boldsymbol{x} = \boldsymbol{B'}\}$. Here $A' \equiv A \texttt{ where } \{\boldsymbol{y} = \boldsymbol{C}\}$ and $B'_i \equiv B_i \texttt{ where } \{\boldsymbol{y} = \boldsymbol{C}\}$.
- (Fixpoint) $\vdash A \texttt{ where } \{x = A\} = x \texttt{ where } \{x = A\}$.
- (Uniqueness) $E = A(E) \vdash E = x \texttt{ where } \{x = A(x)\}$. Here $A(x)$ is a term in which all occurences of the variable x are *guarded*; i.e., all occurrences are inside of set terms.

Recursion inference rule As before, let A and B be terms

$$A \equiv A_0 \texttt{ where } \{x_1 = A_1, \ldots, x_n = A_n\} \quad \text{and}$$
$$B \equiv B_0 \texttt{ where } \{y_1 = B_1, \ldots, y_m = B_m\},$$

with disjoint bound variables and let Σ be a set of equations, each of the form $x_i = y_j$. From $\Delta, \Sigma \vdash A_0 = B_0$ and $\Delta, \Sigma \vdash A_i = B_j$ for each $(x_i = y_j) \in \Sigma$, conclude $\Delta \vdash A = B$.

Set-theoretic axioms

- (Extensionality) Suppose that A and B are both set terms. Then for each $a \in A$,
$$A = B \vdash \{a = b \mid b \in B\}.$$
- ($\bot$) For each set term A, $x \texttt{ where } \{x = x\} = A \vdash$ is an axiom. (Note that the right side of the sequent is the empty set.)

This concludes the list of axioms and inference rules of the proof system for $\mathcal{L}$. As usual, we write $\Delta \vdash \Sigma$ to mean that this judgment can be obtained from the axioms by applications of the inference rules.

The soundness part of the proof is fairly long and tedious due to the notion of groundedness. One can find similar soundness arguments in [6, 7]. Our main result will be a Completeness Theorem for this logic. We prove this in Section 4.3 below. Section 4.1 gives some preliminary results on the deduction system, as does Section 4.2.

However, before turning to those, we mention that the soundness of the Recursion Inference rule is similar to the argument in [7], and we present the soundness argument for the only new axiom in this paper, the Uniqueness axiom.

Soundness of the Uniqueness Axiom. We remind the reader that our official semantics is the one which uses non-wellfounded sets. We shall call upon some results from [3].

Fix formulas $A(x)$ and E, and suppose that α is a substitution with the property that $[\alpha] \|E\| = [\alpha] \|A(E)\|$. Consider the term substitution σ defined on

x alone such that $\sigma(x) = E$. By the Substitution Lemma, $\|A(E)\| = [\hat{\sigma}]\,\|A(x)\|$. We must show that

$$[\alpha]\,\|E\| \quad = \quad [\alpha]\,\|\,x \texttt{ where } \{x = A(x)\}\,\|\,. \tag{16}$$

It will be convenient to assume that the variable x used in x `where` $\{x = A(x)\}$ is completely fresh; i.e., that it does not appear in the finite domain of α, or in the any of the terms in the range of α, or in E. Let z be such a fresh variable. Then $\|\,x \texttt{ where } \{x = A(x)\}\,\| = \|\,z \texttt{ where } \{z = A(z)\}\,\|$. So we may recast (16), using z for x in the recursion term. Our method of showing (16) is to show that both of these are solutions to the same equation, $z = [\alpha]\,\|A(z)\|$. For this, it is essential that z be fresh.

Let e be the substitution defined on z alone so that $e(z) = \|A(z)\|$. Then e has a unique solution, say s. And

$$\|\,z \texttt{ where } \{z = A(z)\}\,\| \quad = \quad s(z) \quad = \quad [s]\,\|A(z)\|\,.$$

Let s' be the substitution with domain z given by $s'(z) = [\alpha]s(z)$. Then the substitution-like operations $[\alpha][s]$ and $[s'][\alpha]$ agree on all variables, hence on all sets built over the variables. (See Exercise 8.3 of [3].) Therefore,

$$\begin{array}{rcl} s'(z) & = & [\alpha]s(z) \\ & = & [\alpha]\,[s]\,\|A(z)\| \\ & = & [s']\,[\alpha]\,\|A(z)\| \end{array}$$

In other words, s' is the solution of the equation we wish to consider, $z = [\alpha]\,\|A(z)\|$.

Next, let τ be the one-point term substitution $z \mapsto E$. Then $\|A(E)\| = [\hat{\tau}]\,\|A(z)\|$. Let t' be the substitution with domain z given by $t'(z) = [\alpha]\hat{\tau}(z)$. As above, $[t']\,[\alpha] = [\alpha]\,[\hat{\tau}]$. So

$$\begin{array}{rcl} t'(z) & = & [\alpha]\,\|A(E)\| \\ & = & [\alpha]\,\|E\| \\ & = & [\alpha]\,[\hat{\tau}]\,\|A(z)\| \\ & = & [t']\,[\alpha]\,\|A(z)\| \end{array}$$

We conclude that t' is also a solution of $z = [\alpha]\,\|A(z)\|$. This last equation only has one solution. (This is where we use the assumption that the original $A(x)$ is guarded in x. If $A(x)$ were just x, for example, then one of the Uniqueness axioms would state that $x = x \models x = \bot$, rendering the system unsound. Also, if x doesn't actually occur in A, the equation $z = [\alpha]\,\|A(z)\|$ still has a unique solution.) So we see that $t'(z) = s'(z)$. This means that

$$[\alpha]\,\|E\| \quad = \quad t'(z) \quad = \quad [\alpha]\,\|\,x \texttt{ where } \{x = A\}\,\|\,,$$

and this verifies that (16) holds.

4.1 Examples and Basic Results

Example 4.1. First, an example to show how the Extensionality and extended Cut rules combine to handle the simple set-theoretic consequences that come up in $\mathcal{L}$. Notice that

$$\begin{array}{l}\{x,y,z\}=\{u,v\}\models\\ \{\{x,y\},\{y,z\},\{z,x\}\}=\{\{x,y\},\{y,z\},\{z,x\},\{x,y,z\}\}.\end{array} \tag{17}$$

Intuitively, if $\{x,y,z\}=\{u,v\}$, then two of the three values x, y, and z must be identical. So $\{x,y,z\}$ must equal one of its doubleton subsets. But can such an entailment be proved in this system? Call the conclusion of (17) ϕ. Replacement applied to the variable w in $E\equiv\{w,x,z\}$ yields $z=y\vdash\{x,z\}=\{x,y,z\}$. A similar instance of Replacement gives $\{x,z\}=\{x,y,z\}\vdash\phi$. Cutting, we have $z=y\vdash\phi$. Similarly, any other equation between two of the variables x,y,z will serve on the left-hand side. Hence, by Extended Cut, it is enough to derive

$$\{x,y,z\}=\{u,v\}\vdash x=z,\ z=y,\ y=x. \tag{18}$$

But now by Extensionality, $\{x,y,z\}=\{u,v\}\vdash x=u, x=v$, and similarly for y and z. And clearly both

$$\begin{array}{ll}x=u,\ y=u,\ z=v\vdash x=z,\ z=y,\ y=x & \text{and}\\ x=v,\ y=u,\ z=v\vdash x=z,\ z=y,\ y=x.\end{array}$$

Hence by Extended Cut,

$$\{x,y,z\}=\{u,v\},\ y=u,\ z=v\vdash x=z,\ z=y,\ y=x.$$

Similarly, $y=v,\ z=v\vdash x=z,\ z=y,\ y=x$, so we can cut on the pair $y=u, y=v$ to get

$$\{x,y,z\}=\{u,v\},z=v\vdash x=z,\ z=y,\ y=x.$$

Finally, we can do the same thing for $z=u$ and then use Extended Cut to eliminate the hypothesis on z altogether. This yields the desired (18).

Example 4.2. Here is another example of how the proof system works. We claim that

$$x=\{x,y\},y=\emptyset,r=\{r,s\},s=\emptyset\models x=r. \tag{19}$$

Here is the proof. The variable x is free in the term $E=\ z$ `where` $\{z=\{z,x\}\}$, and

$$\begin{array}{lcl}[x\mapsto y]E & = & z\ \texttt{where}\ \{z=\{z,y\}\}\\ [x\mapsto s]E & = & z\ \texttt{where}\ \{z=\{z,s\}\}.\end{array}$$

Thus by Replacement

$$y=s\vdash\ z\ \texttt{where}\ \{z=\{z,y\}\}\ =\ z\ \texttt{where}\ \{z=\{z,s\}\}.$$

One of the Uniqueness Axioms is

$$x = \{x, y\} \vdash x = \ z \ \texttt{where} \ \{z = \{z, y\}\},$$

and another is

$$r = \{r, y\} \vdash r = \ z \ \texttt{where} \ \{z = \{z, y\}\}.$$

So by Transitivity, $x = \{x, y\}, r = \{r, y\} \vdash x = r$. To finish the verification of (19), we need only check that

$$x = \{x, y\}, y = \emptyset, r = \{r, s\}, s = \emptyset \vdash r = \{r, y\}, \tag{20}$$

and then use Cut. But (20) follows easily from Replacement.

Note that this example shows that recursive terms might well be needed in the proof of a sequent $\Delta \vdash \phi$, even when the terms in the sequent are explicit.

We conclude this section with several results which are needed for the completeness theorem.

Lemma 4.3. *Suppose that A comes from B by renaming bound variables. Then $A \models B$ and $A \vdash B$.*

Proof. The first assertion is by the Substitution Lemma and some other routine semantic results. The second involves several easy inductions using the Recursion Inference rule.

Next, we have a special case of completeness that plays a key role in the overall result of Section 4.3.

Lemma 4.4. *Let A and B be unquantified terms of $\mathcal{L}$. If $\models A = B$, then $\vdash A = B$.*

Proof. We assume familiarity with the proof of the Simplification Lemma in [6]. By that result, we may assume that

$$\begin{array}{lcl} A & \equiv & x_1 \ \texttt{where} \ \{x_1 = A_1, \ldots, x_n = A_n\} \\ B & \equiv & y_1 \ \texttt{where} \ \{y_1 = B_1, \ldots, y_m = B_m\} \end{array}$$

where each A_i or B_j is a variable, or set of variables. (Essentially, one can repeatedly use the Bekič-Scott axiom to un-nest and combine all the recursions in A and B.) By renaming bound variables, arrange for the lists $\boldsymbol{x}$ and $\boldsymbol{y}$ to be disjoint. Now each of A and B consists essentially of just a flat system of equations. (We are ignoring the $\perp$ to keep things simpler, but this can be added in.) Let s be the solution to A and t the solution to B. We are given that $\|A\| = \|B\|$. So $s(x_1) = t(y_1)$.

There is a bisimulation R such that $R(x_1, y_1)$. Then an application of the Recursion Inference rule with empty Δ derives $\vdash A = B$: the set Σ consists of equations $x_i = y_j$ for all pairs x_i, y_j such that $R(x_i, y_j)$; and the bisimulation property guarantees that $\Sigma \vdash A_i = B_j$ for each equation $(x_i = y_j) \in \Sigma$.

Lemma 4.5. *For $1 \leq i \leq n$, let A_i and B_i be unquantified terms of $\mathcal{L}$. If*

$$\models A_1 = B_1, \ldots, A_n = B_n,$$

then $\vdash A_1 = B_1, \ldots, A_n = B_n$.

Proof. By renaming bound variables if necessary, we may assume that no variable which occurs free in any of the terms also occurs bound. Let the free variables be $X = \{x_1, \ldots, x_k\}$. Let $a_1, \ldots, a_k \in \mathcal{A}$ be distinct atoms which do not occur in any of our terms. Let s be the substitution $x_i \mapsto a_i$. Then $\models [s]A_1 = [s]B_1, \ldots, [s]A_n = [s]B_n$. But the sequent above now has no free variables whatsoever, so the values $[\alpha][s]A_i$ and $[\alpha][s]B_i$ are independent of whichever substitution α we happen to consider. Therefore, for some fixed i, $\models [s]A_i = [s]B_i$.

Now consider s^{-1}. This function is a substitution because its domain is a set of urelements. It is not a term substitution, but it is easy to see that $[s^{-1}]$ does map $\mathcal{L}$ into $\mathcal{L}$. Recall also that we assume that no x_i occurs bound in any of our original terms. Thus, $[s^{-1}][s]A_i = A_i$ and $[s^{-1}][s]B_i = B_i$.

We claim that $\models A_i = B_i$. Let α be any substitution with domain X. Let t be the substitution $a_i \mapsto \alpha(x_i)$, so that $[\alpha][s^{-1}] = [t][\alpha]$. Then

$$[\alpha][s^{-1}][s]A_i \quad = \quad [t][\alpha][s]A_i \quad = \quad [t][\alpha][s]B_i \quad = \quad [\alpha][s^{-1}][s]B_i.$$

Our claim follows. So the overall result follows from Lemma 4.4 and Weakening.

4.2 Analogues of the Deduction Theorem

The lemmas in this section constitute analogues of the Deduction Theorem for the $\mathcal{L}$ logical system. The first is a semantic result, the second syntactic. Before presenting these, we introduce an abbreviation to save on some notation. If σ is a substitution, we write

$$(A = B)\ \texttt{where}\ \{\sigma\} \quad \equiv \quad A\ \texttt{where}\ \{\sigma\} = B\ \texttt{where}\ \{\sigma\}.$$

Note that this is a *formula*, not a term. We also extend this notation from single equations $A = B$ to sets Θ of equations; then $\Theta\ \texttt{where}\ \{\sigma\}$ will abbreviate a set of equations. Note that if Θ is empty, then $\Theta\ \texttt{where}\ \{\sigma\}$ will also be empty.

Lemma 4.6. *Let Θ and Σ be any set of unquantified formulas, and let $\boldsymbol{x}$ be a list of distinct variables. If $\Theta, \boldsymbol{x} = \boldsymbol{D} \models \Sigma$, then*

$$\Theta\ \texttt{where}\ \{\boldsymbol{x} = \boldsymbol{D}\} \models \Sigma\ \texttt{where}\ \{\boldsymbol{x} = \boldsymbol{D}\}. \tag{21}$$

Proof. Let α be any substitution which satisfies every equation in the left-hand side of (21). Let e be the system $x_i = \|D_i\|$, and let s be the solution of e. Let β be the substitution with domain $\{\boldsymbol{x}\} \cup dom(\alpha)$ defined by $\beta(x_i) = [\alpha]s(x_i)$, and

for $y \notin \{\boldsymbol{x}\}$, $\beta(y) = \alpha(y)$. Then $[\beta] = [\alpha]\,[s]$. In particular, $\beta(x_i) = [\alpha]\,[s]\,\|D_i\|$ for all i, so β satisfies $x_i = D_i$. We also claim that β satisfies every equation $E = M$ in Θ. For each such equation,

$$\begin{array}{rcl} [\beta]\,\|E\| & = & [\alpha]\,[s]\,\|E\| \\ & = & [\alpha]\,\|\,E \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}\,\| \\ & = & [\alpha]\,\|\,M \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}\,\| \\ & = & [\beta]\,\|M\| \end{array}$$

So the hypothesis of our lemma applies to β. For some equation $A = B$ in Σ, $[\beta]\,\|A\| = [\beta]\,\|B\|$. And this implies that α satisfies the corresponding equation on the right side of (21), as desired.

Lemma 4.7. *Let Θ be any set of unquantified formulas, and suppose that every occurrence of x in the term D occurs inside a set term. If*

$$\Theta \texttt{ where } \{x = D\} \vdash \Sigma \texttt{ where } \{x = D\}\,,$$

then Θ, $x = D \vdash \Sigma$.

Proof. We want to use the Uniqueness axiom to see that $x = D \vdash x = x \texttt{ where } \{x = D\}$. We may do so because every occurrence of x in D occurs inside a set term.

Let σ be the term substitution $x \mapsto x \texttt{ where } \{x = D\}$. Thus, by Replacement,

$$E = M \vdash [\sigma]E = [\sigma]M$$

for each formula $E = M$ in Θ, provided only that the substitutions are free. Furthermore, the Head axiom is exactly $\vdash [\sigma]E = E \texttt{ where } \{x = D\}$. Hence Θ proves every formula in $\Theta \texttt{ where } \{x = D\}$. Using the hypothesis of our lemma,

$$\Theta \vdash \Sigma \texttt{ where } \{x = D\}\,.$$

Once again, we use the Uniqueness, Replacement, and Head axioms to see that

$$x = D \vdash A \texttt{ where } \{x = D\} = A$$

and similarly for B, for all equations $A = B$ in Σ. Using Transitivity, Extended Cut, and Weakening, we see that Θ, $x = D \vdash \Sigma$.

4.3 Completeness

Our main goal is to prove the following result:

Theorem 4.8 (Completeness/Decidability). *For any finite sets Δ and Γ of formulas of $\mathcal{L}$, $\Delta \vdash \Gamma$ if and only if $\Delta \models \Gamma$. Moreover, this common relation is decidable.*

The soundness of the system was checked at the end of Section 4. Now we turn to completeness. One main idea of the proof is to explore what reasoning can be carried out "within" the head term of a recursion. To this end, it is convenient to extend the our abbreviations from the beginning of Section 4.2. We shall write $\forall \boldsymbol{x}(A = B)$ `where` $\{\sigma\}$ to abbreviate $\forall \boldsymbol{x}(\, A$ `where` $\{\sigma'\} = B$ `where` $\{\sigma'\}\,)$, in which σ' is the same system as σ, but with the equation for the variable x (if any) deleted.

The following lemma shows that we can carry out arbitrary reasoning from the deductive system inside a `where`-context.

Lemma 4.9. *Suppose that $\Delta \vdash \Gamma$, and that σ is any system of equations. Then*

$$\Delta \texttt{ where } \{\sigma\} \vdash \Gamma \texttt{ where } \{\sigma\}\,.$$

Proof. The proof goes by induction on the derivation of $\Delta \vdash \Gamma$. First check that each sequent obtained by applying $\cdot$ `where` $\{\sigma\}$ to all the formulas in an axiom is a derivable sequent. For example, the axiom $A = B \vdash B = A$ becomes

$$A \texttt{ where } \{\sigma\} = B \texttt{ where } \{\sigma\} \vdash B \texttt{ where } \{\sigma\} = A \texttt{ where } \{\sigma\}\,,$$

which is just another instance of the same axiom. Replacement and almost all of the other logical and equational axioms work similarly. Specialization is more complicated. Let σ be any system of equations, and σ' be the same system with the equation for the variable x (if any) deleted. Then we must show that

$$\forall x(\, A(x) \texttt{ where } \{\sigma'\} = B(x) \texttt{ where } \{\sigma'\}\,) \vdash \big(A(E) = B(E)\big) \texttt{ where } \{\sigma\}\,.$$

To do this, Specialize the hypothesis with the term $[x \mapsto x$ `where` $\{\sigma\}\,]E$. Then use Head and Bekič-Scott on the result to yield the desired conclusion.

The $\cdot$ `where` $\{\sigma\}$ analogues of the recursion axioms almost all follow from a trivial application of the Recursion Inference rule with empty Σ. The exception is the Uniqueness axiom, in which it may happen that σ is defined on a variable free in the term E. An example will show how to handle such a case: Suppose that $E \equiv y$, and that σ contains an equation for y. A typical instance of Uniqueness would be:

$$y = \{y\} \vdash y = x \texttt{ where } \{x = \{x\}\}\,.$$

We need to show that

$$(y = \{y\}) \texttt{ where } \{\sigma\} \vdash y \texttt{ where } \{\sigma\} = \big(\, x \texttt{ where } \{x = \{x\}\}\,\big) \texttt{ where } \{\sigma\}\,.$$

Using Head, the right hand side of the new hypothesis equals $\{\, y$ `where` $\{\sigma\}\,\}$; so by Uniqueness, the new hypothesis proves y `where` $\{\sigma\} = x$ `where` $\{x = \{x\}\}$. Since the right-hand side of this does not include free occurrences of y, it may be wrapped in $\cdot$ `where` $\{\sigma\}$ by the Head axiom.

The Bottom axiom carries over trivially. The only remaining axiom is Extensionality. The new hypothesis A `where` $\{\sigma\} = B$ `where` $\{\sigma\}$ in which A and B are both sets of formulas can be converted via the Head axiom into a form

suitable for applying Extensionality; and the results converted back to the form a `where` $\{\sigma\} = b$ `where` $\{\sigma\}$ for $a \in A$ and $b \in B$ by applying Head again, in reverse.

Now we need to check that the $\cdot$ `where` $\{\sigma\}$ form of any sequent derivable by an inference rule is also derivable. There is nothing to show for Weakening and Extended Cut; applying $\cdot$ `where` $\{\sigma\}$ to every formula in either of these rules yields another instance of the same rule. Next, suppose $\Delta \vdash \forall x(\phi(x))$ is derivable by Generalization. In other words, $\Delta \vdash \phi(x)$ and x does not occur free in Δ. Let σ' be the same as σ with x undefined. Then by induction, Δ `where` $\{\sigma'\} \vdash \phi(x)$ `where` $\{\sigma'\}$. By Generalization,

$$\Delta \texttt{ where } \{\sigma'\} \vdash \forall x(\, \phi(x) \texttt{ where } \{\sigma'\}\,) \equiv \big(\forall x(\phi(x))\big) \texttt{ where } \{\sigma\}\,.$$

Since x does not occur free in Δ, by Head and Bekič-Scott Δ `where` $\{\sigma\}$ proves each formula in Δ `where` $\{\sigma'\}$ (the extra condition on x is vacuous). Cutting, Δ `where` $\{\sigma\} \vdash (\forall x(\phi(x)))$ `where` $\{\sigma\}$ as desired.

Finally, suppose that $\Delta \vdash A = B$ by the Recursion inference rule. Then we have that $\Delta, \Sigma \vdash A_0 = B_0$ and $\Delta, \Sigma \vdash A_i = B_j$ for each $(x_i = y_j) \in \Sigma$, and all of these may be established via proofs which are shorter than the given proof. Arrange by change of bound variables that the variables x_i and y_j used in the given instance of Recursion inference do not overlap with the variables in the system σ. This assumption implies that the set of formulas Δ `where` $\{\sigma\}, \Sigma$ proves each formula in (Δ, Σ) `where` $\{\sigma\}$. Hence by induction

$$\Delta \texttt{ where } \{\sigma\}, \Sigma \vdash A_i \texttt{ where } \{\sigma\} = B_j \texttt{ where } \{\sigma\}$$

for each appropriate pair i, j, including $0, 0$. So by Recursion Inference,

$$\Delta \texttt{ where } \{\sigma\} \vdash (\, A_0 \texttt{ where } \{\sigma\}\,) \texttt{ where } \{\ldots, x_i = A_i \texttt{ where } \{\sigma\}, \ldots\} = (\, B_0 \texttt{ where } \{\sigma\}\,) \texttt{ where } \{\ldots, y_j = B_j \texttt{ where } \{\sigma\}, \ldots\}\,.$$

The Bekič-Scott axiom now converts the conclusion of this sequent to the desired $(A = B)$ `where` $\{\sigma\}$.

Finally, we show that universal quantifiers may be eliminated from $\mathcal{L}$.

Lemma 4.10. *For any universally quantified formula $\forall x(\phi(x))$, there is a natural number N_ϕ such that for any fresh atoms $a_1, \ldots, a_{N_\phi} \in \mathcal{A}$,*

$$\phi(a_1), \phi(a_2), \ldots, \phi(a_{N_\phi}) \models \forall x(\phi(x)).$$

Proof. We first consider the possibilities when ϕ is $x = B$ for some term B. If $\|B\| = x$, then $N_\phi = 0$. If $\|B\|$ is some variable other than x, say y, then $N_\phi = 2$, since $a_1 = B$, $a_2 = B$ is unsatisfiable. Finally, if $\|B\|$ is a set term, $N_\phi = 1$ since $a_1 = B$ is unsatisfiable.

Next we consider the case that ϕ is $y = B$ for some variable y distinct from x. If $\|B\| = x$, the previous case applies. If $\|B\|$ is any other variable, then the

universal quantification is vacuous, and $N_\phi = 1$. Similarly, if $\|B\|$ is a set, we may assume that x is an element of the transitive closure of $\|B\|$. Since we are operating in $HF^{1/2}$, the transitive closure of $\|B\|$ is finite, say of cardinality N. We may take $N_\phi = N + 1$. This works because substituting a_i for x cannot increase the cardinality of the transitive closure of a set. Hence, the facts $y = [x \mapsto a_1]\|B\|, \ldots, y = [x \mapsto a_{N+1}]\|B\|$ are inconsistent: any one of them implies that the transitive closure of y has $\leq N$ elements, but taken together they guarantee that each a_i is in the transitive closure of y.

Now we may consider the case that ϕ is $A = B$ where both $\|A\|$ and $\|B\|$ are sets. If there is no substitution for x and the remaining free variables of A and B which satisfies $A = B$, then obviously $N_\phi = 1$ works. Otherwise, consider the finite transitive closures of $\|A\|$ and $\|B\|$. Any substitution satisfying $A = B$ gives rise to a collection of equations between elements of these transitive closures. For example,

$$\{0, x, y, \{0, z\}\} = \{x, y, \{1\}, \{w, 1\}\}$$

might be satisfied by

$$0 = y, x = \{1\}, \{0, z\} = \{w, 1\}, z = 1, w = 0.$$

Of such a list of equations, the ones with a variable on one side suffice to imply that $A = B$; and in order for $A = B$ it is necessary for some such list of equations to hold. There are only finitely many possible such lists of equations, let us say $\Psi_1, \ldots, \Psi_K$. And for each Ψ_i, each equation $\psi \in \Psi_i$ has a corresponding N_ψ by the first two cases, since one side of ψ is a variable. Let N_i be the maximum of all N_ψ for $\psi \in \Psi_i$. Then we may take

$$N_\phi = \sum_{i=1}^{K} N_i.$$

This works because given N_ϕ instances of $A = B$, each one entails one of the K possible lists of equations identified above. Thus, we must have at least N_i instances of some Ψ_i. In particular, we have at least N_ψ instances of each equation in Ψ_i, which entails the universally quantified version of ψ. Thus, the equations in Ψ_i hold for any x, which in turn implies that $A = B$ holds for any x.

All that remains is that case that ϕ itself has universal quantifiers, i.e. $\phi \equiv \forall \boldsymbol{y}(\phi')$ where ϕ' is of the form $A = B$. Then $N_\phi = N_{\phi'}$ works: $N_{\phi'}$ instances of $\forall \boldsymbol{y}(\phi')$ entail for any values of $\boldsymbol{y}$ the corresponding $N_{\phi'}$ instances of ϕ', which in turn entail $\forall x(\phi')$, for those values of $\boldsymbol{y}$. But since the values of $\boldsymbol{y}$ were arbitrary, $\forall x, \boldsymbol{y}(\phi')$ holds, as desired.

Completion of the Proof of Theorem 4.8. We are now equipped to finish the proof of completeness for the overall system. The proof proceeds by reducing the general matter of completeness to successively more special cases, and then the remaining situation will be handled by results which we have already seen. Suppose that $\Delta \models \Gamma$ where Δ and Γ are any finite sets of formulas. First we eliminate universal quantifiers. Suppose that Γ consists of $\{\phi\} \cup \Gamma'$, where ϕ

is of the form $\forall x(\psi(x))$. Choose a fresh variable y. Then $\Delta \models \psi(y), \Gamma'$. Going the other way, if $\Delta \vdash \psi(y), \Gamma'$, then we get $\Delta \vdash \forall y(\psi(y)), \Gamma'$ by Generalization. Lemma 4.3 gets us back to $\Delta \vdash \phi, \Gamma'$. In this way, we have reduced completeness of the entire system to the completeness of the system for sequents whose conclusions contain only unquantified equations $A = B$ between terms.

Similarly, we may reduce completeness to the case where there are no universal quantifiers to the left of $\vdash$. Suppose that Δ contains a quantified formula $\forall x(\delta(x))$. Let Δ' be the same as Δ, but with $\forall \boldsymbol{x}(\delta(\boldsymbol{x}))$ replaced by

$$\delta(a_1), \ldots, \delta(a_{N_\delta}).$$

By the previous lemma, $\Delta' \models \forall x(\delta(x))$, and hence it entails each formula in Δ, so $\Delta' \models \Gamma$. And assuming $\Delta' \vdash \Gamma$, it is trivial to use Specialization to see that $\Delta \vdash \Gamma$. Proceeding by induction, we may assume that Δ also contains no universal quantifiers.

Next, by suitable changes of variables bound by `where`, arrange that no bound variable occurs freely anywhere nor occurs in more than one term of Δ or Γ. We can now eliminate an instance of recursion in Δ: replace the formula

$$C = (\, D_0 \texttt{ where } \{x_1 = D_1, x_2 = D_2, \ldots, x_n = D_n\}\,)$$

(if such a formula belongs to Δ) by the set of formulas

$$C = D_0,\ x_1 = D_1',\ x_2 = D_2',\ \ldots,\ x_n = D_n'$$

where D_i' is D_i if x_i is a grounded variable and $D_i' = x \texttt{ where } \{x = x\}$ if x_i is ungrounded. Call the resulting set of formulas Δ'. Again, Δ' semantically entails each formula in Δ because the solution to the system of equations $\{x_i = D_i\}$ is unique by AFA. Now suppose that $\Delta' \vdash \Gamma$. By Lemma 4.9, $\Delta' \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\} \vdash \Gamma \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}$. But it is easy to see that Δ proves each formula in $\Delta' \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}$. And each formula $(A = B) \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}$ in $\Gamma \texttt{ where } \{\boldsymbol{x} = \boldsymbol{D}\}$ proves the corresponding $A = B$ in Γ by the Head axiom, since the $\boldsymbol{x}$ do not occur in A or B. Hence $\Delta \vdash \Gamma$.

The argument of the last paragraph shows how to remove the outermost recursion from one term in Δ. Using it repeatedly, we reduce to the case where Δ is recursion-free. That is, we need only consider the case in which Δ is a finite set of equations between explicit terms. Thus, every term occurring in Δ is either a bare variable, or a set of variables. Our next goal is to show that it suffices to consider only equations in which at least one side is a bare variable. So suppose that Δ contains some equation of the form $C = D$ where C and D are both sets of terms, $C = \{c_1, \ldots, c_n\}$, $D = \{d_1, \ldots, d_m\}$. We may assume that $n \geq m$. For each surjective map ν from $\{1, \ldots n\}$ onto $\{1, \ldots m\}$, consider the set of formulas Δ_ν which consists of Δ with $C = D$ replaced by the set of formulas $c_i = d_{\nu(i)}$ for each i from 1 to n. Then each Δ_ν semantically entails each formula in Δ, and hence $\Delta_\nu \models \Gamma$. On the other hand, if each $\Delta_\nu \vdash \Gamma$, then we use Extensionality to patch these all back together to show that $\Delta \vdash \Gamma$ in a fashion similar to Example 4.1 above.

Thus, all that remains is the case where each equation in Δ has a bare variable on (say) the left-hand side. We next want to ensure that no variable occurs on the left-hand side of more than one equation in Δ. So, suppose that $x = C$ and $x = D$ are both elements of Δ. It may happen that $x = C$ is provable from the remaining elements of Δ. If so, then it may be removed from Δ without harm. If not, replace $x = C$ by $C = D$ to produce Δ'. Note that $\Delta' \vdash x = C$, and hence $\Delta' \models \Gamma$; moreover, if $\Delta' \vdash \Gamma$, then $\Delta \vdash \Gamma$, since $\Delta \vdash C = D$. This change to Δ may have re-introduced equations which do not have a bare variable on either side. Apply the previous reduction step to Δ' again. Continue in this fashion so long as the set of hypotheses contains two equations with the same variable on the left-hand side. This process must terminate, because any equation $y = E$ that is removed from Δ at any stage is provable from the set of hypotheses at any later stage. Thus, $y = E$ will never have to be added again. Moreover, only subterms of terms occurring in the original Δ will ever be used, and there are only finitely many of these.

Thus, we may suppose that Δ is a set of equations, each of the from $x = D_x$ where x is a variable, D_x is a term, and no variable occurs on the left-hand side of more than one equation. In this way, Δ is essentially a term substitution. Furthermore, all equations of the form $x = x$ are instances of the first equational axiom and may be cut away. So no D_x is identically x. Moreover, if D_x is any other variable, say y, then we may eliminate the variable x altogether by substitution: $[x \mapsto y](\Delta) \models [x \mapsto y]\Gamma$; and if $[x \mapsto y](\Delta) \vdash [x \mapsto y]\Gamma$, then $\Delta \vdash \Gamma$ since Δ proves every formula in $[x \mapsto y](\Delta)$ and for all $A \in \mathcal{L}$, $\Delta \vdash A = [x \mapsto y]A$ by Replacement via $x = y$. Thus, we may assume that the right-hand side of every equation in Δ is not a bare variable; i.e., it is a set.

We have reduced matters to the case of a semantic assertion of the form

$$x_1 = A_1, \ldots, x_n = A_n \models \Gamma.$$

By Lemma 4.6,

$$\models \Gamma \texttt{ where } \{x_1 = A_1, \ldots, x_n = A_n\}.$$

By Lemma 4.5,

$$\vdash \Gamma \texttt{ where } \{x_1 = A_1, \ldots, x_n = A_n\}.$$

And then by an iterated use of Lemma 4.7 and the Bekič-Scott Axiom, we see that

$$x_1 = A_1, \ldots, x_n = A_n \vdash \Gamma.$$

This concludes the proof of the completeness portion of Theorem 4.8. To see the decidability, note that converting a term to a simplified (flat) form is effective, deciding whether flat systems are bisimilar is decidable, as is deciding whether a variable in a flat system is grounded or not. Further, note that each of the reductions in the proof of completeness is computable. Formally, the decision procedure would be recursive, since in our reduction step we needed to know whether a particular equation was provable from some set of assumptions. However, each needed instance of the decision problem involves formulas smaller than those in the original sequent.

5 Conclusion

The main point of this paper has been to present a language for reasoning about fixed-point terms involving sets, to present a logical system for the equality of such terms, and finally to prove the soundness and completeness of the logical system. The predecessors of this work include many papers on completeness results for logics of recursion, beginning with [4] and also including [6] and [7]. However, the semantics in these papers are all much more general than what we study in this paper. This means that the restricted setting of the paper has made more logical laws valid. Specifically, the Uniqueness axiom and also Extensionality are new here. Even more, the logical system in this paper is a sequent calculus, and this is the first time that such a calculus has been used this area.

We have used the interpretation via non-wellfounded sets as the primary one in the paper. However, if one prefers, there is always the option of interpreting fixed point terms via finite pointed graphs modulo bisimulation, as we did in Section 3.1. One reason for opting for the set semantics is that many of the semantic principles needed in proving the soundness of the system have already been shown, mainly in [3]. So this has shortened our presentation. In the other direction, one can use the more categorical semantics of Section 3.3. But here there would be more work to do, since the results in this paper are specific to sets and the power set functor.

We close with a few questions which we have not pursued in this paper but which we feel should be interesting:

- The universal quantifier turned out to be eliminable in $\mathcal{L}$. So it might be interesting to extend the syntax to also allow existential quantification.
- $\mathcal{L}$ could also be extended to allow infinitary formulas and sets of equations. Is the system still complete for the appropriate interpretation?
- The language $\mathcal{L}$ is only rudimentary when it comes to set theory, being basically the language of pointed graphs. It should be interesting to enrich the language with operations such as union or powerset, or with the membership relation, and then look for matching complete deductive systems. (Such complete systems will of course not exist for sufficiently powerful variants of $\mathcal{L}$.)
- A final project would be to take seriously the more general semantics which we presented in Section 3.3 and to present a logical calculus for fixed-point terms interpreted on initial iterative algebras of finitary set functors.

References

1. Aczel, P.: Non-Well-Founded Sets. CSLI Lecture Notes Number, vol. 14. CSLI Publications, Stanford (1988)
2. Adámek, J., Milius, S., Velebil, J.: Iterative Algebras at Work. Math. Structures Comput. Sci. 16(6), 1085–1131 (2006)

3. Barwise, J., Moss, L.S.: Vicious Circles. CSLI Lecture Notes, vol. 60. CSLI Publications, Stanford (1996)
4. Courcelle, B., Kahn, G., Vuillemin, J.: Algorithmes d'equivalence et de reduction a des expressions minimales dans une classe d'equations recursives simples. In: Loeckx, J. (ed.) ICALP 1974. LNCS, vol. 14, pp. 200–213. Springer, Heidelberg (1974)
5. Forti, M., Honsell, F.: Set Theory with Free Construction Principles. Ann. Scuola Norm. Sup. Pisa Cl. Sci. (4) 10(3), 493–522 (1983)
6. Hurkens, A.J.C., McArthur, M., Moschovakis, Y.N., Moss, L.S., Whitney, G.: The Logic of Recursive Equations. Journal of Symbolic Logic 63(2), 451–478 (1998)
7. Moss, L.S.: Recursion and Corecursion Have the Same Equational Logic. Theoretical Computer Science 294(1-2), 233–267 (2003)
8. Nelson, E.: Iterative Algebras. Theoretical Computer Science 25(1), 67–94 (1983)

Overloading Is NP-Complete

A Tutorial Dedicated to Dexter Kozen

Jens Palsberg

UCLA, University of California, Los Angeles

Abstract. We show that overloading is NP-complete. This solves exercise 6.25 in the 1986 version of the Dragon book.

1 Introduction

Overloading is a form of polymorphism in which a name denotes multiple functions and "the context is used to decide which function is denoted by a particular instance of the name" [3]. Many programming languages support overloading. For example, in MATLAB [8] the name `mpower` is overloaded to denote both:

- a function of two arguments `a,b` where `a` is a square matrix and the exponent `b` is a scalar, and
- a function of two arguments `a,b` where `a` is a scalar and the exponent `b` is a square matrix.

When we call `mpower` in MATLAB, the arguments will be used to decide which function will be called. Both functions return a square matrix. Similarly, in Java we can program an interface with two methods that are both named `mpower`:

```
interface Math {
  SquareMatrix mpower(SquareMatrix a, Scalar b);
  SquareMatrix mpower(Scalar a, SquareMatrix b);
}
```

If a class implements `Math` and we call `mpower` on an object of that class, the arguments will be used to decide which method will be called.

We say that an implementation *resolves overloading* when it decides which overloaded function will be called.

How does a language implementation decide which function will be called? The easiest case is when different functions have different numbers of arguments. In that case, the number of arguments at the call site is sufficient to decide which function will be called. The harder case is when different functions have the same number of arguments, like in the `mpower` example above. In that case, the *types* of the arguments at the call site must be used to decide which function will be called.

Both MATLAB and Java has a notion of type associated with each value. A MATLAB value can be a square matrix or scalar, for example, and a Java value

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 204–218, 2012.

can be a SquareMatrix object or a Scalar object, for example. The type of a value determines which operations can be performed on that value. For example, we can perform `mpower` on a scalar and a square matrix, but not on two square matrices.

MATLAB and Java differ in *when* the type of a value is known to the implementation. MATLAB has a *dynamic* type system in which the types are known at run time but are unknown to the compiler, at least in some cases. So, in some cases, a MATLAB implementation must decide at run time which overloaded function to call:

> "MATLAB determines which implementation to call at the time of evaluation in this case." [12]

Java has a *static* type system in which the type of every value is known to the compiler. Java requires that a Java implementation can resolve overloading by investigating the number and the types of the arguments passed into a method:

> "When a method is invoked, the number of actual arguments (and any explicit type arguments) and the compile-time types of the arguments are used, at compile time, to determine the signature of the method that will be invoked This enables a Java implementation to decide at compile time which overloaded function to call." [7]

For dynamically typed languages like MATLAB, overloading may decrease runtime performance. This happens particularly when MATLAB has to examine the types of function arguments at run time to decide which overloaded function to call.

For statically typed languages like Java, overloading is essentially a syntactic convenience that frees programmers from inventing different names for functions with similar functionality. Instead, the responsibility to invent such different names is passed on to the compiler. The compiler uses the types of function arguments to do a source-to-source transformation that eliminates the use of overloading by giving different names to different functions. After such a transformation, compilation can proceed as usual.

In this paper we focus on the computational complexity of overloading resolution in statically typed languages. In Sections 2–7 we will explain in detail why overloading resolution is NP-complete for an important case: a λ-calculus with overloading. Our proof consists of three polynomial-time reductions:

$$\begin{aligned} \text{3SAT} \;&\leq\; \text{Monotone One-in-Three 3SAT} \;\leq\; \text{Overloading Resolution} \\ &\leq\; \text{Constraint Solving} \end{aligned}$$

and an easy proof that the Constraint Solving problem is in NP. Thus, all of the listed problems are NP-complete. In Section 8 we will discuss interactions of overloading and other language features.

Our proof is inspired by two lines of research. The first inspiration is hardware description languages that allow component overloading [17,11]. The idea

of component overloading is similar to function overloading: the context is used to decide which component is denoted by a particular instance of a name. Vacharajani et al [17], and Mathaikutty and Shukla [11] both sketched reductions that show that component overloading is NP-complete. Our reduction targets a λ-calculus with overloading and we give a detailed proof.

The second inspiration is a paper by Kozen et al. [10] who showed how to reduce a type-checking problem to a constraint solving problem.

In the 1986 version of the Dragon book [1], Exercise 6.25 is to show that overloading is NP-complete:

> ****6.25** The resolution of overloading becomes more difficult if identifier declarations are optional. More precisely, suppose that declarations can be used to overload identifiers representing function symbols, but that all occurrences of an undeclared identifier have the same type. Show that the problem of determining if an expression in this language has a valid type is NP-complete. This problem arises during type checking in the experimental language Hope (Burstall, MacQueen, and Sannella [1980]). [1, p.384]

However, the exercise is difficult, we think, and the literature contains no detailed, formal proof, hence this tutorial. We formalize the exercise as a problem about a λ-calculus, as we explain next.

2 Example Language

Our example language is a λ-calculus with overloading:

$$(\textit{Expression}) \qquad e ::= x \mid \lambda x.e \mid e\ e$$

We use x to range over identifiers. Overloaded functions all come from the initial environment: each free variable of an expression refers to an overloaded function. We assume that in every expression, the bound variables are distinct and different from the free variables.

Each overloaded function has a restricted form of *intersection type* [5,9]: each intersection type is an intersection of simple types. When a program uses one of the overloaded functions from the initial environment, the function gets one of the simple types in the intersection. In contrast, every programmer-defined function has a simple type.

Our interpretation of Exercise 6.25 in the 1986 version of the Dragon book is that the initial environment provides identifiers that are declared with intersection types, while each programmer-defined function leaves an identifier undeclared, that is, without a type annotation, and required to have a simple type.

Let us now define the type system formally. We use c to range over a finite set of at least two base types. The types are:

$$\begin{aligned} (\textit{Type}) \quad s, t &::= c \mid t \to t \\ (\textit{Intersection Type}) \quad u &::= u \wedge u \mid t \end{aligned}$$

If $u = (\ldots \wedge t \wedge \ldots)$, then we write $u \leq t$.

A type environments is a finite mapping from identifiers to types of the form u. We use A to range over type environments.

A type judgment is of the form $A \vdash e : t$. The type rules are:

$$A \vdash x : t \qquad \text{if } A(x) \leq t \tag{1}$$

$$\frac{A[x : s] \vdash e : t}{A \vdash \lambda x.e : s \to t} \tag{2}$$

$$\frac{A \vdash e_1 : s \to t \qquad A \vdash e_2 : s}{A \vdash e_1\ e_2 : t} \tag{3}$$

We say that an expression e is typable with a type environment A if and only if there exists t such that $A \vdash e : t$ is derivable.

We define the resolution of overloading decision problem as follows:

Overloading Resolution
Instance: An expression e and a type environment A.
Problem: Is e typable with A?

Lemma 1. *Suppose a derivation of $A \vdash e_0 : t$ contains the judgment $A' \vdash e' : t'$. If x is a free variable of e_0, then $A(x) = A'(x)$.*

Proof. We proceed by induction on e_0. We have three cases.

- If $e_0 \equiv x$, then e' is also x, so we have $A = A'$, hence $A(x) = A'(x)$.
- If $e_0 \equiv \lambda y.e$, then from the induction hypothesis and type rule (2), we have (1) $(A[y : s])(x) = A'(x)$. From (1) and that bound variables are different from the free variables, we have $A(x) = (A[y : s])(x) = A'(x)$, as desired.
- If $e_0 \equiv e_1 e_2$, then e' occurs in either e_1 or e_2. Let us do a case analysis of the two cases. Suppose e' occurs in e_1. From the induction hypothesis and type rule (3), we have $A(x) = A'(x)$, as desired. The other case where e' occurs in e_2 is similar, and also here we immediately get $A(x) = A'(x)$, as desired. □

3 Constraints

We define

$$(\mathit{Term}) \quad r ::= v \mid c \mid r \to r$$

A constraint system over a set of type variables V is a finite collection of constraints of the forms:

$$u \leq r$$
$$r = r'$$

where u is an intersection type as defined above, and each variable that occurs in r or r' is a member of V.

We use φ to range over finite mappings from type variables to types of the form t.

We define:

$$\varphi(c) = c$$
$$\varphi(r_1 \to r_2) = \varphi(r_1) \to \varphi(r_2)$$

A constraint system C has solution φ if and only if

- for each constraint $u \leq r$ in C, we have $u \leq \varphi(r)$,
- for each constraint $r = r'$ in C, we have $\varphi(r) = \varphi(r')$, and

We say that a constraint system C is satisfiable if C has a solution.

Constraint Solving
Instance: A constraint system C.
Problem: Is C satisfiable?

Theorem 1. *Constraint Solving is in NP.*

Proof. Let C be a constraint system. For each constraint $u \leq r$ in C, where $u = \bigwedge_{i=1}^{n} t_i$, guess t_{i_0} and replace constraint $u \leq r$ with the constraint $t_{i_0} = r$. The resulting constraint system can be solved with first-order unification which is doable in polynomial time [13]. □

Let us define a transformation S on constraint systems. The idea of S is to remove a constraint without changing whether the constraint system is satisfiable.

We define the transformation S:

$$S(C, v = r) = \begin{cases} (C \setminus \{\ v = r\ \})[v := r] & \text{if } (v = r) \in C \text{ and } v \text{ doesn't occur in } r \\ C & \text{otherwise} \end{cases}$$

Intuitively, $S(C, v = r)$ removes the constraint $v = r$ from C and then replaces all occurrences of v by r in the resulting constraint system.

Lemma 2. *C is satisfiable if and only if $S(C, v = r)$ is satisfiable.*

Proof. We have two cases.

If $C = S(C, v = r)$, then the lemma is immediate.

If C contains the constraint $v = r$, where v doesn't occur in r, then we consider the two directions of the lemma.

In the forwards direction, we have immediately that if C has solution φ, then also $S(C, v = r)$ has solution φ.

In the backwards direction, suppose $S(C, v = r)$ has solution φ. We now have two cases.

In the first case, suppose v doesn't occur in $C \setminus \{\ v = r\ \}$. Let $\{\ v_1, \ldots, v_m\ \}$ be the set of type variables that occur in r but don't occur in $C \setminus \{\ v = r\ \}$. Let c be a base type. Define:

$$\psi = \varphi,\ (v \mapsto (\varphi(r))[v_1 := c, \ldots, v_m := c]),\ (v_1 \mapsto c),\ \ldots,\ (v_m \mapsto c)$$

We have immediately that ψ solves all constraints in $C \setminus \{\ v = r\ \}$. For the constraint $v = r$, we have that $\psi(v) = (\varphi(r))[v_1 := c, \ldots, v_m := c] = \psi(r)$ so ψ also solves $v = r$.

In the second case, suppose v does occur in $C \setminus \{\ v = r\ \}$. Notice that since v occurs in $C \setminus \{\ v = r\ \}$, we have that r occurs in $(C \setminus \{\ v = r\ \})[v := r]$. Define:

$$\psi = \varphi,\ (v \mapsto \varphi(r))$$

We have immediately that ψ solves all constraints in $C \setminus \{\ v = r\ \}$. For the constraint $v = r$, we have that $\psi(v) = \varphi(r) = \psi(r)$, so ψ also solves $v = r$. □

4 From Overloading to Constraints

We will now show a reduction of the overloading resolution problem to a constraint solving problem. The reduction is useful both for showing that overloading resolution is in NP and that it is NP-hard.

For a expression e_0 and a type environment A, we define the set V_{e_0} of type variables:

$$\begin{aligned} V_{e_0} = &\{\ v_x \mid x \text{ is an occurrence of a free variable in } e_0\ \} \\ &\cup \{\ v_x \mid \lambda x.e \text{ is an occurrence of a subexpression in } e_0\ \} \\ &\cup \{\ v_{\lambda x.e} \mid \lambda x.e \text{ is an occurrence of a subexpression in } e_0\ \} \\ &\cup \{\ v_{e_1 e_2} \mid e_1 e_2 \text{ is an occurrence of a subexpression in } e_0\ \} \end{aligned}$$

From e_0 and A, generate these type constraints over V_{e_0}:

- For each occurrence of a free variable x in e_0, the constraint $A(x) \leq v_x$.
- For each occurrence of $\lambda x.e$ in e_0, the constraint $v_{\lambda x.e} = v_x \to v_e$.
- For each occurrence of $e_1 e_2$ in e_0, the constraint $v_{e_1} = v_{e_2} \to v_{e_1 e_2}$.

We use $C_{e_0,A}$ to denote the constraint system generated from e_0 and A.

Theorem 2. *An expression e_0 is typable with type environment A if and only if $C_{e_0,A}$ is satisfiable.*

Proof. For the forwards direction, assume that e_0 is typable with type environment A. In other words, there exists t_0 such that $A \vdash e_0 : t_0$ is derivable.

We will define a function $\varphi : V_{e_0} \to \mathit{Type}$. In the derivation of $A \vdash e_0 : t$, each occurrence of a subexpression e' of e_0 occurs exactly once in a type judgment of the form $A' \vdash e' : t'$. If the occurrence of e' is a free variable of e_0 or of one of the forms $\lambda x.e$ and $e_1 e_2$, then define $\varphi(v_{e'}) = t'$. Additionally, each occurrence of

a subexpression $\lambda x.e'$ of e_0 occurs exactly once in a type judgment of the form $A' \vdash \lambda x.e' : s' \to t'$; define $\varphi(v_x) = s'$.

We need to show that $C_{e_0,A}$ has solution φ. Let us do a case analysis on the members of $C_{e_0,A}$.

- For an occurrence of a free variable x in e_0, we have the constraint $A(x) \leq v_x$. From the type rule (1) and Lemma 1, we have that there exists a type t such that $A(x) \leq t$ and $\varphi(v_x) = t$. We conclude that φ solves the constraint.
- For an occurrence of $\lambda x.e$ in e_0, we have the constraint $v_{\lambda x.e} = v_x \to v_e$. From the type rule (2), we have that there exist types s, t such that $\varphi(v_{\lambda x.e}) = s \to t$ and $\varphi(v_x) = s$ and $\varphi(v_e) = t$. We conclude that φ solves the constraint.
- For an occurrence of $e_1 e_2$ in e_0, we have the constraint $v_{e_1} = v_{e_2} \to v_{e_1 e_2}$. From the type rule (3), we have that there exist types s, t such that $\varphi(v_{e_1}) = s \to t$ and $\varphi(v_{e_2}) = s$ and $\varphi(v_{e_1 e_2}) = t$. We conclude that φ solves the constraint.

This concludes the proof of the forwards direction.

For the backwards direction, assume that $C_{e_0,A}$ is satisfiable. Let φ a solution of $C_{e_0,A}$. For each occurrence of a subexpression e' of e_0, let $x_1, \ldots, x_n$ be the bound variables of the λ-abstractions that enclose e' in e_0. Define

$$A_{e'} = A, (x_1 : \varphi(v_{x_1})), \ldots, (x_n : \varphi(v_{x_n}))$$

Notice that $A = A_{e_0}$. We will prove that for each occurrence of a subexpression e' of e_0, we have

$$A_{e'} \vdash e' : \varphi(v_{e'})$$

We proceed by induction on e'. We have four cases.

- For an occurrence of a free variable x in e_0, we have (1) $A(x) \leq \varphi(v_x)$. Notice that (2) $A(x) = A_x(x)$. From (1), (2), we have $A_x(x) = A(x) \leq \varphi(v_x)$ so we can use type rule (1) to conclude $A_x \vdash x : \varphi(v_x)$, as desired.
- For an occurrence of a bound variable x in e_0, we have from type rule (1) that $A_x \vdash x : \varphi(v_x)$, as desired.
- For an occurrence of $\lambda x.e$ in e_0, we have (1) $\varphi(v_{\lambda x.e}) = \varphi(v_x) \to \varphi(v_e)$. From the induction hypothesis used on e, we have (2) $A_e \vdash e : \varphi(v_e)$. Next notice that (3) $A_e = A_{\lambda x.e}, (x : \varphi(v_x))$. From (2), (3), and type rule (2), we conclude (4) $A_{\lambda x.e} \vdash \lambda x.e : \varphi(v_x) \to \varphi(v_e)$. From (1) and (4), we conclude $A_{\lambda x.e} \vdash \lambda x.e : \varphi(v_{\lambda x.e})$, as desired.
- For an occurrence of $e_1 e_2$ in e_0, we have (1) $\varphi(v_{e_1}) = \varphi(v_{e_2}) \to \varphi(v_{e_1 e_2})$. From the induction hypothesis used on e_1 and e_2, we have (2) $A_{e_1} \vdash e_1 : \varphi(v_{e_1})$ and $A_{e_2} \vdash e_2 : \varphi(v_{e_2})$. Next notice that (3) $A_{e_1} = A_{e_2} = A_{e_1 e_2}$. From (2) and (3) we conclude (4) $A_{e_1 e_2} \vdash e_1 : \varphi(v_{e_1})$ and $A_{e_1 e_2} \vdash e_2 : \varphi(v_{e_2})$. From type rule (3) and from (1) and (4), we derive $A_{e_1 e_2} \vdash e_1 e_2 : \varphi(v_{e_1 e_2})$, as desired.

This concludes the proof of the backwards direction. □

5 Monotone One-in-Three 3SAT

The Monotone One-in-Three 3SAT problem is an excellent fit for proving that overloading resolution is NP-hard. Schaefer proved that Monotone One-in-Three 3SAT is NP-complete [14]. Indeed, he proved a more general result with a proof that is a bit complicated. We will give a straightforward proof that Monotone One-in-Three 3SAT is NP-complete.

We will use 0,1 to denote the Boolean values *false*, *true*, respectively.

Let R be a three-place Boolean relation which is true if and only if exactly one of its three arguments is true. Thus, $R(1,0,0) = R(0,1,0) = R(0,0,1) = 1$, while $R(1,1,1) = R(1,1,0) = R(1,0,1) = R(0,1,1) = R(0,0,0) = 0$.

We use φ to range over mappings from variables to Boolean values.

We say that a mapping φ *satisfies* a formula if, after we replace each variable x with $\varphi(x)$, the formula evaluates to 1. We also say that a formula is *satisfiable* if there exists a mapping φ that satisfies the formula.

We define the function T:

$$T(z_1, z_2, z_3) = R(z_1, a_1, a_4) \wedge R(z_2, a_2, a_4) \wedge R(a_1, a_2, a_5) \wedge R(a_3, a_4, a_6) \wedge R(z_3, a_3, f) \wedge R(t, f, f)$$

where in each application of T we have that $a_1, a_2, a_3, a_4, a_5, a_6, t, f$ are fresh and distinct variables.

We say that φ' extends φ if and only if $dom(\varphi') \supseteq dom(\varphi) \wedge \forall x \in dom(\varphi') \cap dom(\varphi) : \varphi'(x) = \varphi(x)$.

Lemma 3. *Let φ be an assignment of the variables z_1, z_2, z_3 to Boolean values. We have that φ satisfies $(z_1 \vee z_2 \vee z_3)$ if and only if φ can be extended to φ' that satisfies $T(z_1, z_2, z_3)$.*

Proof. In the forwards direction, let us extend each of the seven mappings that satisfy $(z_1 \vee z_2 \vee z_3)$ to mappings that also satisfy $T(z_1, z_2, z_3)$:

Mapping	z_1	z_2	z_3	a_1	a_2	a_3	a_4	a_5	a_6	t	f
φ'_1	0	0	1	0	0	0	1	1	0	1	0
φ'_2	0	1	0	1	0	1	0	0	0	1	0
φ'_3	0	1	1	1	0	0	0	0	1	1	0
φ'_4	1	0	0	0	1	1	0	0	0	1	0
φ'_5	1	0	1	0	1	0	0	0	1	1	0
φ'_6	1	1	0	0	0	1	0	1	0	1	0
φ'_7	1	1	1	0	0	0	0	1	1	1	0

In the backwards direction, consider the only mapping 1) $\varphi = [z_1 \mapsto 0, z_2 \mapsto 0, z_3 \mapsto 0]$ that doesn't satisfy $(z_1 \vee z_2 \vee z_3)$. Suppose we can extend φ to φ' that satisfies $T(z_1, z_2, z_3)$. From $R(t, f, f)$ we have that $\varphi'(t) = 1$ and 2) $\varphi'(f) = 0$. From $R(z_3, a_3, f)$, (1), and (2), we have 3) $\varphi'(a_3) = 1$. From $R(a_3, a_4, a_6)$ and (3), we have 4) $\varphi'(a_4) = 0$. From $R(z_1, a_1, a_4) \wedge R(z_2, a_2, a_4)$ and (1) and (4), we have 5) $\varphi'(a_1) = \varphi'(a_2) = 1$. However, (5) implies that φ' doesn't satisfy $R(a_1, a_2, a_5)$, a contradiction. □

Let us now define the Monotone One-in-Three 3SAT problem.

Monotone One-in-Three 3SAT
Instance: A formula $\bigwedge_{i=1}^{n} R(x_{i1}, x_{i2}, x_{i3})$, where each x_{ij} is a variable.
Problem: Is the formula satisfiable?

Theorem 3. *Monotone One-in-Three 3SAT is NP-complete.*

Proof. Monotone One-in-Three 3SAT is in NP because we can guess a mapping φ and then check in polynomial time where φ satisfies the formula.

To prove that Monotone One-in-Three 3SAT is NP-hard we will show a reduction from 3SAT. An instance of 3SAT is a formula of the form $\mathcal{F} = \bigwedge_{i=1}^{n}(l_{i1} \vee l_{i2} \vee l_{i3})$, where each l_{ij} is either a variable x_i or the negation of a variable $\overline{x}_i$. The 3SAT problem is whether the formula is satisfiable. 3SAT is NP-complete [6].

Let $x_1, \ldots, x_m$ be the variables used in $\mathcal{F}$. We will define a formula $\mathcal{H}$ over the variables $x_1, \ldots, x_m, y_1, \ldots y_m, t, f$, where $y_1, \ldots y_m, t, f$ are all distinct and different from $x_1, \ldots, x_m$. We will use the helper mapping π:

$$\pi(x_i) = x_i$$
$$\pi(\overline{x}_i) = y_i$$

Now we define $\mathcal{H}$:

$$\mathcal{H} = [\ \bigwedge_{i=1}^{n} T(\pi(l_{i1}), \pi(l_{i2}), \pi(l_{i3}))\ ]\ \ \wedge\ \ [\ \bigwedge_{j=1}^{m} R(x_j, y_j, f)\ ]\ \ \wedge\ \ R(t, f, f)$$

Notice that $\mathcal{H}$ is an instance of Monotone One-in-Three 3SAT. Notice also that, for each $j \in 1..m$, the clauses $R(x_j, y_j, f_1)$ and $R(t, f, f)$ force any assignment that satisfies $\mathcal{H}$ to map x_j to 1 and y_j to 0, or map x_j to 0 and y_j to 1. The reason is that $R(t, f, f)$ forces the assignment to map f to 0, and so the assignment must map exactly one of x_j and y_j to 1. So, y_j plays the role of $\overline{x}_j$.

Suppose φ satisfies $\mathcal{F}$. From Lemma 3 we have that we can extend φ to φ' such that φ' satisfies $T(\pi(l_{i1}), \pi(l_{i2}), \pi(l_{i3}))$. From the observation above about y_j, t, f we have that we can easily extend φ' to φ'' such that φ'' satisfies $\mathcal{H}$.

Conversely, suppose φ satisfies $\mathcal{H}$. From Lemma 3 we have that φ satisfies $\mathcal{F}$. □

6 From Monotone One-in-Three 3SAT to Overloading

Let F, T be two base types. Define

$$u_0 = (T \to F \to F \to T) \wedge (F \to T \to F \to T) \wedge (F \to F \to T \to T)$$

For an instance of Monotone One-in-Three 3SAT

$$\mathcal{H} = \bigwedge_{i=1}^{n} R(x_{i1}, x_{i2}, x_{i3})$$

that uses the variables $y_1, \ldots, y_m$. Thus, each x_{ij} is one of $y_1, \ldots, y_m$. We define the type environment:

$$A = \{\ (f_1 \mapsto u_0), \ldots,\ (f_n \mapsto u_0)\ \}$$

and we define the λ-expression:

$$e_0 = \lambda g.\lambda y_1 \ldots . \lambda y_m.\ g\ (f_1 x_{11} x_{12} x_{13})\ \ldots\ (f_n x_{n1} x_{n2} x_{n3})$$

Theorem 4. *$\mathcal{H}$ is satisfiable if and only if e_0 is typable with a type environment A.*

Proof. From Theorem 2 we have that e_0 is typable with a type environment A if and only if $C_{e_0,A}$ is satisfiable. So all we need to prove is that:

$$\mathcal{H} \text{ is satisfiable if and only if } C_{e_0,A} \text{ is satisfiable.} \tag{4}$$

Let us calculate $C_{e_0,A}$:

$$v_{\lambda g....} = v_g \to v_{\lambda y_1....} \tag{5}$$
$$v_{\lambda y_1....} = v_{y_1} \to v_{\lambda y_2....} \tag{6}$$
$$\ldots$$
$$v_{\lambda y_m....} = v_{y_m} \to v_{g\ (f_1 x_{11} x_{12} x_{13})\ \ldots\ (f_n x_{n1} x_{n2} x_{n3})} \tag{7}$$
$$v_{g\ (f_1 x_{11} x_{12} x_{13})\ \ldots\ (f_{n-1} x_{(n-1)1} x_{(n-1)2} x_{(n-1)3})} = v_{f_n x_{n1} x_{n2} x_{n3}} \to v_{g\ (f_1 x_{11} x_{12} x_{13})\ \ldots\ (f_n x_{n1} x_{n2} x_{n3})} \tag{8}$$
$$\ldots$$
$$v_{g\ (f_1 x_{11} x_{12} x_{13})} = v_{f_2 x_{21} x_{22} x_{23}} \to v_{g\ (f_1 x_{11} x_{12} x_{13})\ (f_2 x_{21} x_{22} x_{23})} \tag{9}$$
$$v_g = v_{f_1 x_{11} x_{12} x_{13}} \to v_{g\ (f_1 x_{11} x_{12} x_{13})} \tag{10}$$
$$v_{f_i x_{i1} x_{i2}} = v_{x_{i3}} \to v_{f_i x_{i1} x_{i2} x_{i3}} \quad i \in 1..n \tag{11}$$
$$v_{f_i x_{i1}} = v_{x_{i2}} \to v_{f_i x_{i1} x_{i2}} \quad i \in 1..n \tag{12}$$
$$v_{f_i} = v_{x_{i1}} \to v_{f_i x_{i1}} \quad i \in 1..n \tag{13}$$
$$u_0 \leq v_{f_i} \quad i \in 1..n \tag{14}$$

Notice that for each constraint in the lines (5)–(13), the left-hand side doesn't occur on the right-hand side. So, we can use the S transformation $(m+4n)$ times on those constraints to produce the following constraint system that we call C':

$$u_0 \leq v_{x_{i1}} \to v_{x_{i2}} \to v_{x_{i3}} \to v_{f_i x_{i1} x_{i2} x_{i3}} \quad i \in 1..n$$

From Lemma 2 we have that each of the $(m + 4n)$ applications of S preserves satisfiability so we have:

$$C_{e_0,A} \text{ is satisfiable if and only if } C' \text{ is satisfiable} \tag{15}$$

We can combine (4) and (15) and get that we must prove:

$$\mathcal{H} \text{ is satisfiable if and only if } C' \text{ is satisfiable} \tag{16}$$

In the forwards direction, suppose $\mathcal{H}$ has solution φ. We will use the helper function δ:

$$\delta(1) = T$$
$$\delta(0) = F$$

Define:

$$\psi(v_{x_{ij}}) = \delta(\varphi(x_{ij})) \qquad i \in 1..n,\ j \in 1..3$$
$$\psi(v_{f_i x_{i1} x_{i2} x_{i3}}) = T$$

From that $\mathcal{H}$ has solution φ, we have that for each $i \in 1..n$, exactly one of $\varphi(x_{ij}), j \in 1..3$ is 1, while the other two are 0. So for each $i \in 1..n$, exactly one of $\delta(\varphi(x_{ij})), j \in 1..3$ is T, while the other two are F. We conclude that ψ solves C'.

In the backwards direction, suppose C' has solution ψ. We will use δ^{-1} to denote the inverse of δ. Define:

$$\varphi(x_{ij}) = \delta^{-1}(\psi(v_{x_{ij}})), \qquad i \in 1..n,\ j \in 1..3$$

From C' and the definition of u_0, we have that for each $i \in 1..n$, exactly one of $\psi(v_{x_{ij}}), j \in 1..3$, is T, while the other two are F. So for each $i \in 1..n$, exactly one of $\delta^{-1}(\psi(v_{x_{ij}})), j \in 1..3$, is 1, while the other two are 0. We conclude that φ solves $\mathcal{H}$. □

7 Putting It All Together

Theorem 5. *Overloading Resolution and Constraint Solving are both NP-complete.*

Proof. From Theorem 3 we have that (1) Monotone One-in-Three 3SAT is NP-complete. From (1) and Theorem 4, we have that (2) Overloading Resolution is NP-hard. From (2) and Theorem 2, we have that (3) Constraint Solving is NP-hard. From Theorem 1 we have that (4) Constraint Solving is in NP. From (3) and (4) we have that (5) Constraint Solving is NP-complete. From (5) and Theorem 2 we have that (6) Overloading Resolution is in NP. From (2) and (6) we have that Overloading Resolution is NP-complete.

8 Interactions of Overloading and Other Language Features

While a compiler can resolve overloading for all practical statically typed languages of which we are aware, the complexity of the resolution algorithm varies from language to language. The complexity varies because overloading may be restricted in various ways and because overloading may interact with other language constructs. Let us consider some of the possibilities.

8.1 Type Annotations versus Type Inference

Our example language relies on type inference to assign a type to every bound variable. The essence of our *overloading is NP-complete* theorem is that such type inference must be done by exhaustive search. We have implemented such a search algorithm for a subset of MATLAB and we found that it works well in practice.

In contrast to our λ-calculus, Java requires every formal parameter to be annotated with a type. This changes the complexity of the overloading problem from NP-complete to polynomial time. To see this, let us first take a look at what our reduction from Monotone One-in-Three 3SAT to Overloading might look like if we target Java instead of λ-calculus. For example, we might map the formula:

$$R(x_1, x_2, x_3)$$

to this Java program:

```
class B { }
class T implements B { }
class F implements B { }

public class Test {
  T f(T a, F b, F c) { return new T(); }
  T f(F a, T b, F c) { return new T(); }
  T f(F a, F b, T c) { return new T(); }

  T run() {
    B x1,x2,x3;
    return f(x1,x2,x3);
  }
}
```

The idea of the Java program is as follows. Class `B` is a common superclass of two classes `T` and `F` that represent the Boolean values. In class `Test`, the three versions of the overloaded method `f` mimic the intersection type of f in the λ-calculus program. In the `run` method, we don't know what we need to store in the variables `x1,x2,x3` to satisfy the formula so we declare them to be of type `B`. The expression `f(x1,x2,x3)` is the same kind of call that we used in the λ-term.

The Java program doesn't type check! The problem lies with the type annotation `B` for the variables `x1,x2,x3`. The type annotation prevents the expression `f(x1,x2,x3)` from type checking because none of the declared `f` methods take an argument of type `B`. If only we could omit the annotation `B`, then one could imagine that type inference could assign a type `T` or `F` to each of `x1,x2,x3`. However, Java doesn't support such type inference so this style of reduction to Java doesn't work.

Let us now explain why overloading resolution in Java can be done in polynomial time. We will do an informal proof by induction on the structure of expressions. In the base case, we have expressions such as variables and constants

whose nonoverloaded types are known to the compiler from type annotations or the language specification. Java allows overloading only of methods so in the key induction step, we can consider a call of an overloaded method. From the induction hypothesis we have that the compiler knows a nonoverloaded type for each argument expression. Recall the quote from Section 1 that says that Java resolves overloading by using the number and compile-time types of the arguments. Thus the compiler can now either declare the call type correct or give a type error. This concludes the informal proof for Java.

Notice that the proof relies on type annotations for variables, which is exactly what ruined the Java example above.

8.2 Overloading and Subtyping

Java has all of overloading, type annotations, and subtyping. Java's notion of subtyping presents no new problems for overloading resolution. The reason is that Java's type system enables the compiler to know a static type of every expression, even in the presence of subtyping. Thus, the informal induction proof in the previous subsection works for Java in this case too. We conclude that also in the presence of subtyping can we resolve overloading in Java in polynomial time.

Castagna, Ghelli, and Longo [4] studied a typed λ-calculus in which the programmer can define overloaded functions. This goes beyond the λ-calculus in Section 2 where all overloaded functions all come from the initial environment. As far as we know, the problem of devising a type checker or a type inference algorithm for their calculus remains an open problem.

8.3 Overloading and Hindley-Milner Polymorphism

Let us consider the notion of polymorphism known as Hindley-Milner polymorphism that can be found in such languages as ML and Haskell. If we combine overloading with Hindley-Milner polymorphism, the result is an undecidable type system [15,20,16]. The undecidability result has led researchers to look for restrictions of overloading. The idea is that Hindley-Milner polymorphism together with restricted overloading may be decidable. Volpano showed that if we in a system with Hindley-Milner polymorphism make a Haskell-style restriction on overloading, the resulting type system is NP-hard [18,19]. Camarao, Figueiredo, and Vasconcellos studied another restriction on overloading and reported on experiments with a type checking algorithm [2].

9 Conclusion

We have given a detailed proof of why overloading resolution is NP-complete for a typed λ-calculus. We hope that the proof techniques will be helpful to researchers who want to prove similar results for other languages. We also hope that our paper can help clarify which overloading problems are NP-complete and

which problems have higher complexity due to interactions with other language features.

Exercise 1: Design a variant of our calculus that captures the essence of overloading in Java. Prove that type checking can be done in polynomial time.

Exercise 2: Consider a variant of our calculus in which every bound variable is annotated with a simple type, using the notation $\lambda x : t.e$. What is the complexity of type checking?

Exercise 3: Consider a variant of our calculus in which we disallow $\lambda x.e$. What is the complexity of type checking?

Acknowledgments. We thank Matt Brown, Jakob Rehof, and Alexander Sherstov for helpful comments and discussions.

References

1. Aho, A.V., Sethi, R.I., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Addison-Wesley, Reading (1986)
2. Camarao, C., Figueiredo, L., Vasconcellos, C.: Constraint-set satisfiability for overloading. In: Proceedings of PPDP (2004)
3. Cardelli, L., Wegner, P.: On understanding types, data abstraction, and polymorphism. ACM Computing Surveys 17(4), 471–522 (1985)
4. Castagna, G., Ghelli, G., Longo, G.: A calculus for overloaded functions with subtyping. Information and Computation 117(1), 115–135 (1995)
5. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Principal type schemes and lambda-calculus semantics. In: Seldin, J., Hindley, J. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 535–560. Academic Press (1980)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman (1979)
7. Gosling, J., Joy, B., Steele, G., Bracha, G.: Java Language Specification. 3rd edn. (2005), `http://java.sun.com/docs/books/jls/third_edition/html/classes.html#8.4.9`
8. Higham, D.J., Higham, N.J.: MATLAB Guide. 2nd edn. SIAM (2005)
9. Roger Hindley, J.: Types with intersection: An introduction. Formal Aspects of Computing 4, 470–486 (1991)
10. Kozen, D., Palsberg, J., Schwartzbach, M.I.: Efficient inference of partial types. Journal of Computer and System Sciences 49(2), 306–324 (1994); Preliminary version in Proceedings of FOCS 1992
11. Mathaikutty, D.A., Shukla, S.K.: Metamodeling-Driven IP Reuse for SoC Integration and Microprocessor Design. Artech House (2009)
12. MathWorks. Product documentation, matlab compiler (2011), `http://www.mathworks.com/help/techdoc/matlab_prog/f2-38133.html`
13. Paterson, M.S., Wegman, M.N.: Linear unification. Journal of Computer and System Sciences 16, 158–167 (1978)
14. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1995), pp. 216–226 (1978)

15. Smith, G.: Polymorphic Type Inference for Languages with Overloading and Subtyping. PhD thesis, Cornell University (1991)
16. Smith, G.S.: Principal type schemes for functional programs with overloading and subtyping. Science of Computer Programming 23, 197–226 (1994)
17. Vachharajani, M., Vachharajani, N., Malik, S., August, D.I.: Facilitating reuse in hardware models with enhanced type inference. In: Proceedings of IEEE/ACM/IFIP Second International Conference on Hardware/Software Codesign and System Synthesis (1994)
18. Volpano, D.: Haskell-style overloading is NP-hard. In: Proceedings of ICCL 1994, Fifth IEEE International Conference on Computer Languages, pp. 88–95 (May 1994)
19. Volpano, D.: Lower bounds on type checking overloading. Information Processing Letters 57, 9–14 (1996)
20. Volpano, D., Smith, G.: On the Complexity of ML Typability with Overloading. In: Hughes, J. (ed.) FPCA 1991. LNCS, vol. 523, pp. 15–28. Springer, Heidelberg (1991)

Combining Epistemic Logic and Hennessy-Milner Logic

Sophia Knight[1], Radu Mardare[2], and Prakash Panangaden[3]

[1] Ecole Polytechnique and INRIA Saclay
[2] Department of Computer Science, Aalborg University
[3] School of Computer Science, McGill University

Abstract. We define an epistemic logic for labelled transition systems by introducing equivalence relations for the agents on the states of the labelled transition system. The idea is that agents observe the dynamics of the system modulo their ability to distinguish states and in the process learn about the current state and past history of the execution. This is in the spirit of dynamic epistemic logic but is a direct combination of Hennessy-Milner logic and epistemic logic. We give an axiomatization for the logic and prove a completeness theorem with respect to the class of models obtained by unfolding labelled transition systems.

Dedicated to Dexter Kozen on the occasion of his 60th birthday.

1 Introduction

Dexter Kozen was one of the pioneers of logic and computation. Among his numerous and varied contributions to the subject was an early joint paper with Rohit Parikh [1] where they established an elementary proof of the completeness of the Segerberg axioms for propositional dynamic logic (PDL) simplifying an earlier proof of Parikh. One of us, Prakash Panangaden, learned basic modal logic – and many other things – from Dexter when he arrived at Cornell as a professor in 1985. The elegance of his presentation and the confident way in which he blasted through all obstacles on the way to establishing a proof had a profound influence on Panangaden. Over a quarter of a century later, this paper on a completeness proof for a logic that combines Hennessy-Milner-van Bentham logic with epistemic logic, bears the imprint of Dexter's masterful presentation of the intricacies of the completeness proof of PDL.

Concurrency theory has been built upon the implicit assumption of omniscience of all the agents involved, but for many purposes – notably security applications – it is crucial to incorporate and reason about what agents "know" or do not know. Tracking the flow of information is the essence of analyses of security protocols. Equally crucial is the idea that different participants may have different views of the system and hence know different things. The purpose of this paper is to meld traditional concurrency concepts with epistemic concepts and define a logic with both dynamic and epistemic modalities.

Epistemic logic has been a major theme within distributed systems ever since the groundbreaking paper of Halpern and Moses [2], but has been strangely

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 219–243, 2012.

slow to influence concurrency theory. A few investigations have appeared but, as far as we know, there has not been a thorough integration of epistemic concepts with the traditional theory of labelled transition systems. Typically one sees a multimodal logic closely tied to the syntax of some particular process calculus with reasoning principles that are not proven complete in any sense [3]. Such logics are interesting and useful but their close tie to a particular process formalism obscures the general principles. Another closely related strand is, of course, dynamic epistemic logic [4] which, as the name makes manifest, is all about how knowledge evolves. However, the bulk of this work is about actions that communicate information, perhaps through messages or announcements, rather than about general transitions that could change basic facts about the state. A few papers indeed deal with so-called fact-changing actions but, as far as we know, the theory is still geared toward communication actions. Our goals are to develop the theory for a suitable general class of labelled transition systems and to formulate axioms that are provably complete with respect to this class of models. We provide more detailed comparisons with related work in a later section after the presentation of our framework.

The standard route to modelling epistemic concepts is to use Kripke models: these are sets of states equipped with indistinguishability (equivalence) relations [5]. We will equip the states with a labelled transition system structure as well and impose coherence conditions between the two kinds of relations. The resulting modal logic is a blend of Hennessy-Milner logic, epistemic logic and temporal modalities. The essential point is that one can reason about how knowledge changes as transitions occur. There are many variations that one could contemplate and the particular formalism that we have developed is geared toward representing the unfolding of a labelled transition system through time taking into account different agents' differing views of the labelled transition system.

The paper is organized as follows. In the next section we review background material on labelled transition systems and Hennessy-Milner logic. In Section 3 we define the class of transition systems that we work with; they are called *history labelled transition systems* and are unfoldings of the usual labelled transition systems, with the addition of equivalence relations on states. In Section 4 we define the logic and its semantics. In Section 5 we prove the weak completeness theorem. There is an easy argument, which we present in Section 5, that shows that a strong completeness theorem is not possible. The final sections discuss related work and conclusions.

2 Background

We assume familiarity with basic concepts like labelled transition systems (LTSs) and epistemic logic. For the benefit of readers who may not be familiar with these ideas we give a brief overview in this section. An excellent exposition of general modal logics is the text book by Blackburn et al. [6] called *Modal Logic*.

Definition 1. *A* labelled transition system *is a triple* $(S, \mathcal{A}, \rightarrow \subseteq S \times \mathcal{A} \times S)$, *where* S *is a set of* states, $\mathcal{A}$ *is a set of* actions *and* $\rightarrow$ *is a* labelled transition relation. *We will write* $s \xrightarrow{a} s'$ *when* $(s, a, s') \in \rightarrow$.

The idea is that S represents the possible states of a dynamical system. The system can perform certain actions and these cause a change in the state. The resulting state is not completely determined by the initial state and the action so that one has a transition relation rather than a function. Some actions may not be possible in some states, if an action a is possible from state s we say that a is *enabled* in s.

There are various senses in which states may be deemed to be equivalent. A canonical one is called *bisimulation*. The idea of bisimulation is that if the actions possible from two states *and all of their successors* do not distinguish them, they should be deemed equivalent. Here is a formal definition.

Definition 2. *We say that an equivalence relation* R *on the state space* S *of an LTS is* a bisimulation relation *if whenever* sRt *and* $s \xrightarrow{a} s'$ *then there exists some* t' *such that* $t \xrightarrow{a} t'$ *with* $s'Rt'$. *We say that* s *and* t *are* bisimilar *if there is some bisimulation relation relating them.*

Since R is required to be an equivalence relation it follows that the analogous condition holds with the roles of s and t exchanged. The properties of bisimulation are discussed at length in the concurrency theory literature, see, for example [7,8] or in the modal logic literature, see, for example [9].

There is a remarkable theorem due independently to van Benthem and to Hennessy and Milner that gives a modal characterization of bisimulation. The logic has come to be called Hennessy-Milner logic. The basic constructs are the boolean connectives and a modal operator written $\langle a \rangle$ or its dual $[a]$, where the a's appearing in the formulas are actions associated with the LTS being studied. The definition of satisfaction for these formulas follows the standard inductive construction due to Tarski with only the modal operator requiring explicit explanation. This is given by $s \models \langle a \rangle \phi$ iff $s \xrightarrow{a} s'$ and $s' \models \phi$. The fundamental theorem is the following.

Theorem 3. *Assume that* $(S, \mathcal{A}, \rightarrow)$ *is a labelled transition system with the property that for a given* s *and* a *the set of* s' *such that* $s \xrightarrow{a} s'$ *is finite*[1]. *Then two states* s *and* t *are bisimilar iff they satisfy all the same formulas of Hennessy-Milner logic.*

The basic setup for modelling epistemic logic is due to Kripke [10]; see *Reasoning About Knowledge* by Fagin et al. [5]. Consider the set S of possible states[2] of some system. We have a finite set of *agents*, typically written $\mathcal{I} = \{i, j, \ldots\}$. We define a modal operator – one for each agent – written K_i. The idea is that the formula $K_i\phi$ means that the agent i *knows* the fact ϕ. The axioms usually used are due to Hintikka [11]:

[1] Such systems are said to be *image finite*.

[2] They are often called *possible worlds* in the philosophical literature.

0.	All propositional tautologies.	
1.	$K_i\phi \Rightarrow \phi$	Only truths can be known.
2.	$K_i(\phi \Rightarrow \psi) \Rightarrow (K_i\phi \Rightarrow K_i\psi)$	Deductive closure.
3.	$K_i\phi \Rightarrow K_iK_i\phi$	Positive introspection.
4.	$\neg K_i\phi \Rightarrow K_i\neg K_i\phi$	Negative introspection.

These are used together with the following rules of inference.

$$\frac{\phi}{K_i\phi} \qquad\qquad \frac{\phi \quad \phi \Rightarrow \psi}{\psi}$$

The semantics for this logic is given in terms of *indistinguishability relations*. The idea is that a particular agent has only limited awareness of everything that might be true in a state. In particular, an agent might not be able to distinguish two states. We associate with each agent an equivalence relation that models its ability to distinguish two states.

Definition 4. *A* Kripke structure *is a set S of states, a finite set $\mathcal{I}$ of agents, a set $\mathcal{P}$ of primitive propositions, for each state s a set $\pi(s) \subset \mathcal{P}$ and for each $i \in \mathcal{I}$ an equivalence relation $\sim_i$ on S.*

The meaning of an atomic proposition is built into the definition of the Kripke structure: $s \models p$ iff $p \in \pi(s)$; the meaning of the boolean connectives is standard. We define the meaning of the modal formula as follows: $s \models K_i\phi$ iff for every state s' such that $s \sim_i s'$, $s' \models \phi$. The fundamental completeness theorem is that a formula is provable from the Hintikka axioms if and only if it is true in all Kripke structures.

3 Histories

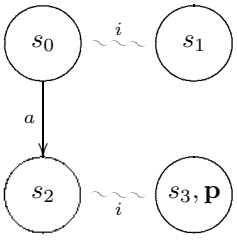
The main contribution of this paper is to study how an agent's knowledge changes as transitions occur in a labelled transition system. The basic picture is that the agent has a limited view of the states of the labelled transition system and this is modelled by an equivalence relation on the states of the system just as in a Kripke structure. The agent does not choose the actions to perform but can see which action has happened and tries to deduce from this where it is. Our temporal-epistemic logic will be designed to handle this type of reasoning.

The semantics of the formulas will be given in terms of histories or runs, as with the semantics of Halpern and Moses [2,12], but we view the runs as coming from the executions of a labelled transition system (LTS). In fact, we will view the set of runs as forming a labelled transition system in its own right. This will give a "branching-time" logic rather than a linear-time logic. We will use the box and diamond modalities of Hennessy-Milner logic [13] rather than the "always" and "eventually" modalities of temporal logic. In this section, we motivate the need for this particular combination of modalities.

The basic set up for a purely epistemic (static) logic is a set of states with equivalence relations, one for each agent. If we wish to incorporate this into a given labelled transition system the natural step is to define equivalence relations on the

states of the labelled transition system. If one does this naïvely one gets situations where one cannot say what an agent has learned from its history.

Example 1. Consider the following simple labelled transition system:



where the wiggly line refers to the indistinguishability equivalence relation of agent i and the proposition $\mathbf{p}$ holds in the state s_3 and in no other state. The agent i in state s_0 cannot tell whether he is in s_0 or in s_1. Similarly, in s_2 he cannot tell whether he is in s_2 or in s_3. However, if the agent is in s_0 and then observes an a action then he "knows" he must have been in s_0 and further, that he is in s_2 now. No purely state-based semantics can say this. It is only because the agent "remembers" how he got there that one can say anything. Thus, a purely state based semantics is not adequate for even the simplest statements about evolving knowledge for agents with memory and basic reasoning abilities. □

The basic paradigm that we have in mind is that the agent is observing a transition system: the agent can see the actions and can remember the actions but cannot *control* the actions nor see *which actions are available at a given state*. The extent to which an agent can "see" the state is what the indistinguishability relation spells out.

In order to give the semantics of the epistemic modalities we need to extend the equivalence relation from states to histories. We formalize labelled transition systems, histories and this equivalence relation as follows.

Definition 5. *An **epistemic labelled transition system** is a set of states, S, a finite set of actions $\mathcal{A}$, and, for every $a \in \mathcal{A}$, a binary relation, written $\xrightarrow{a}$, on the states. We write $s \xrightarrow{a} s'$ instead of $(s, s') \in \xrightarrow{a}$. In addition, there is a finite set of **agents**, denoted by letters like $i, j, \ldots$. For each agent i there is an equivalence relation, written $\sim_i$ defined on S.*

The relation $\xrightarrow{a}$ can be nondeterministic and does not have to be image-finite [3]. From now on we always mean an epistemic labelled transition system when we use the phrase "labelled transition system." We also assume that all actions are visible, that is, there are no hidden actions (commonly denoted by τ).

Definition 6. *A **history** is a finite alternating sequence of states and actions*

$$s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n,$$

where, for each $l \in \{0, \ldots, n-1\}$, $s_l \xrightarrow{a_{l+1}} s_{l+1}$.

[3] "Image finite" means that for a given s and a the set $\{s' | s \xrightarrow{a} s'\}$ is finite.

Given a pair of histories, an agent can tell immediately that they are not the same if they do not have exactly the same action. In order to say this it will be convenient to define the notation $\mathsf{act}(h)$ to mean the action sequence extracted from the history h; it has an evident inductive definition. Given a history h, we write $h[n]$ for the n^{th} state in h. Thus if $h = s_0a_1s_1a_2s_2a_3s_3$, $\mathsf{act}(h) = a_1a_2a_3$ and $h[0] = s_0$ while $h[2] = s_2$. We write $|h|$ for the length of the sequence of states in h.

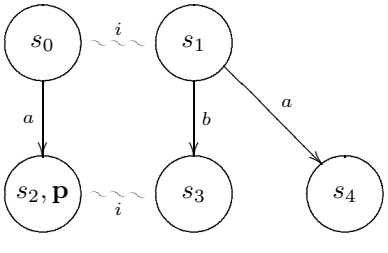
Definition 7. *We say that the histories h_1 and h_2 are* ***indistinguishable by agent*** $\mathbf{i}$*, written $h_1 \sim_i h_2$, if: (i) $\mathsf{act}(h_1) = \mathsf{act}(h_2)$ and (ii) for all $0 \leq n \leq |h_1|(= |h_2|)$, $h_1[n] \sim_i h_2[n]$.*

The use of the same notation for indistinguishability of states and histories should not occasion anxiety for the reader as the context will disambiguate which we mean; this usage is meant to emphasize the tight connection between the concepts.

It is useful to have both past and future modalities. We will define the syntax precisely in the next section, for the moment we note that $\langle - \rangle$ means one step in the past and $\langle + \rangle_a$ means *possibly* after an a-step into the future (we will see later why the future operator is concerned with possibility while the past operator is not). Consider the labelled transition system we have used for our example above. Suppose we introduce the proposition $@s$ to mean "at the state s" then we want to be able to say things like $s_0as_2 \models K_i\langle - \rangle @s_0$. Note that we cannot say $s_0 \models K_i@s_0$, so we need the past operator to express the idea that agent i learns where he was in the past, or, in general, learns that a fact used to be true. Note that, for this example, $s_0as_2 \models \langle - \rangle K_i@s_0$ does not hold, even though $s_0as_2 \models K_i\langle - \rangle @s_0$ does.

Note that every history has a beginning and every state has a finite number of predecessors: in short the prefix order on histories is well founded. This will cause most of the difficulties in the completeness proof.

Example 2. Why do we need the Hennessy-Milner like modalities indexed by actions? Consider the following simple labelled transition system:



which is like the previous example except for the addition of the extra state and transitions and the fact that $\mathbf{p}$ is true in s_2 instead of s_3. We would like to be able to say $s_0 \models \langle + \rangle_a K_i\mathbf{p}$. Note that s_4 can be distinguished by i from any other state. Without the ability to label the diamonds with a we would have to write $s_0 \models \langle + \rangle K_i\mathbf{p}$ which is simply mot true. The point is that the agents can see the

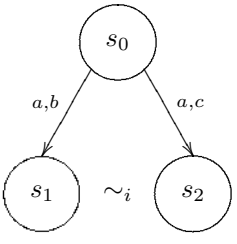
labels on the transitions and use them to gain knowledge; in order to describe this the action labels must be on the Hennessy-Milner modalities. □

The logic, though its semantics is given in terms of runs, is actually a branching time logic. It is applied to a very specific type of transition system that arises as the set of histories of general labelled transition system. The "states" are histories and the transitions are of the form

$$s_0 a_1 s_1 \ldots a_n s_n \xrightarrow{a} s_0 a_1 s_1 \ldots a_n s_n a s$$

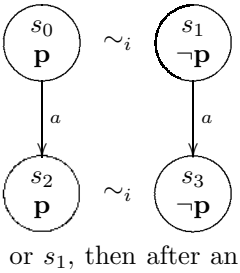
whenever $s_n \xrightarrow{a} s$ is a transition of the underlying labelled transition system. The key features of these labelled transition systems of histories are: a well foundedness property for the backward transitions, determinacy for the backward transitions and a few other properties.[4] In the course of the completeness proof we will spell out these properties and then proceed with the axiomatization and completeness theorem.

Example 3. Here is an example about why the identity of actions is important.



If this system starts out in s_0 and an a action occurs, then agent i will not know which state he is in, because s_1 and s_2 are equivalent for the agent. But if the system does a b action, then the agent knows he is in s_1 because he observes the b action and s_1 is the only state that a b action leads to. Similarly, if the system does a c action, then the agent knows that the system is in s_2. □

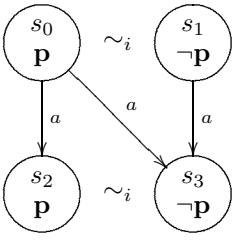
Example 4. This example shows why we want to be able to combine epistemic modalities and (past or future) temporal modalities. Here p represents some proposition.



If the system starts out in s_0 or s_1, then after an a action, the agent does not know whether p is true, but he does know that if p is true now, then it must have been true in the first state, and if p is false now, it must have been false in the first state. □

[4] In fact, such transition systems arise naturally as unfoldings of general labelled transition systems.

Example 5



If this system starts out in s_0 or s_1 and then an a action occurs, then after the action, the agent does not know whether p is true, but he knows that if p is true now, then it was true in the start state. But he also knows that if p is not true now, then p may or may not have been true in the start state. □

3.1 History Systems

First we will explain how to translate any LTS with equivalence classes into an equivalent history LTS: an LTS with designated starting states, where the entire history of any run starting from a starting state is determined by its current state.

Definition 8. *Given the LTS* $(S_0, \mathcal{A}, \mathcal{I}, \underset{0}{\longrightarrow}, \sim^0)$*, where* S_0 *is the set of states,* $\mathcal{A}$ *is the set of actions,* $\mathcal{I}$ *the set of agents,* $\underset{0}{\longrightarrow} \subseteq S_0 \times \mathcal{A} \times S_0$ *is the transition relation and* $\sim^0 \subseteq S_0 \times \mathcal{I} \times S_0$ *is the indistinguishability relation, inductively construct the* unfolding $(S_1, \mathcal{A}, \mathcal{I}, \underset{+}{\longrightarrow}, \underset{-}{\longrightarrow}, \underset{+}{\overset{*}{\longrightarrow}}, \underset{-}{\overset{*}{\longrightarrow}}, \sim^1)$*, where* $\underset{+}{\longrightarrow} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\underset{-}{\longrightarrow} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\underset{+}{\overset{*}{\longrightarrow}} \subseteq S_1 \times S_1$ *and* $\underset{-}{\overset{*}{\longrightarrow}} \subseteq S_1 \times S_1$*, as follows:*

1. *If* $s \in S_0$ *then* $s \in S_1$.
2. *If* $s_0.a_1.s_1.a_2...s_n \in S_1$ *and* $s_n \underset{0}{\overset{a}{\longrightarrow}} s$ *then* $s_0.a_1...s_n.a.s \in S_1$ *and* $s_0.a_1...s_n \underset{+}{\overset{a}{\longrightarrow}} s_0.a_1...s_n.a.s$.
3. *If* $s_0.a_1...s_n$, $s_0.a_1...s_n.a.s \in S_1$ *then* $s_0.a_1...s_n.a.s \underset{-}{\overset{a}{\longrightarrow}} s_0.a_1...s_n$.
4. *If* $s_0.a_1...s_n \in S_1$ *then* $s_0.a_1...s_n \underset{+}{\overset{*}{\longrightarrow}} s_0.a_1...s_n$.
5. *If* $s_0.a_1...s_n$, $s_0.a_1...s_n.a_{n+1}...a.s \in S_0$ *then* $s_0.a_1...s_n \underset{+}{\overset{*}{\longrightarrow}} s_0.a_1...s_n.a_{n+1}...a.s$.
6. *If* $s_0.a_1...s_n \in S_1$ *then* $s_0.a_1...s_n \underset{-}{\overset{*}{\longrightarrow}} s_0.a_1...s_n$.
7. *If* $s_0.a_1...s_n$, $s_0.a_1...s_n.a_{n+1}...a.s \in S_0$ *then* $s_0.a_1...s_n.a_{n+1}...a.s \underset{-}{\overset{*}{\longrightarrow}} s_0.a_1...s_n$.
8. *If* $s, t \in S_0$ *and* $s \sim_i^0 t$ *then* $s \sim_i^1 t$.
9. *If* $s, t \in S_1$ *and* $s \sim_i^1 t$ *and* $s \underset{+}{\overset{a}{\longrightarrow}} s.a.s'$ *and* $t \underset{+}{\overset{a}{\longrightarrow}} t.a.t'$ *and* $s' \sim_i^0 t'$ *then* $s.a.s' \sim_i^1 t.a.t'$.

Definition 9. *An LTS with agent equivalence classes and with transition relations* $\underset{+}{\longrightarrow} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\underset{-}{\longrightarrow} \subseteq S_1 \times \mathcal{A} \times S_1$*,* $\underset{+}{\overset{*}{\longrightarrow}} \subseteq S_1 \times S_1$ *and* $\underset{-}{\overset{*}{\longrightarrow}} \subseteq S_1 \times S_1$ *is called a* history-LTS *if it satisfies the following properties:*

1. *Forward and backward transitions are converse:* $s \xrightarrow[+]{a} t$ *iff* $t \xrightarrow[-]{a} s$.
2. *There is only one way to reach each state: if* $s \xrightarrow[+]{a} t$ *then for all states* s' *and all actions* b, *if* $s' \xrightarrow[+]{b} t$ *then* $s = s'$ *and* $a = b$.
3. *If we let* $\xrightarrow[+]{} = \bigcup_{a \in \mathcal{A}} \xrightarrow[+]{a}$, *then* $\xrightarrow[+]{*}$ *is the transitive reflexive closure of* $\xrightarrow[+]{}$.
4. *If we let* $\xrightarrow[-]{} = \bigcup_{a \in \mathcal{A}} \xrightarrow[-]{a}$, *then* $\xrightarrow[-]{*}$ *is the transitive reflexive closure of* $\xrightarrow[-]{}$.
5. *There are no infinite backward paths: it is impossible to have an infinite chain* $s_0 \xrightarrow[-]{} s_1 \xrightarrow[-]{} \ldots \xrightarrow[-]{} s_n \xrightarrow[-]{} \ldots$.
6. $\sim_i$ *is transitive, reflexive and symmetric for each agent* i.
7. *If* $s_1 \sim_i t_1$ *and there exists a state* s_0 *and an action* a *such that* $s_0 \xrightarrow[+]{a} s_1$ *then there exists a state* t_0 *such that* $t_0 \xrightarrow[+]{a} t_1$ *and* $s_0 \sim_i t_0$.

These properties capture the idea that a history LTS is exactly what we get when we unfold the paths of an LTS with agent equivalence relations; a formal proof is straightforward. At each stage there is possible future branching but the past is determined in a particular history. Thus the past modalities are like LTL modalities but not the future modalities. The starred modalities give one the power of "always" and "eventually" operators in temporal logics. A history is assumed to have a starting point so it must be well founded.

4 The Logic and Its Semantics

In this section we present the logic. It allows us to discuss what is true at a certain state, what was true in the past, what agents know at at the current state, and what may or must be true in the future.

We assume a finite set of agents $\mathcal{I}$, a finite set of actions $\mathcal{A}$, and a countable set of propositions Q. In the following definition, $a \in \mathcal{A}$, $i \in \mathcal{I}$, and $q \in Q$.

Definition 10 (Syntax)

$$\phi \ := \ \top \mid q \mid \langle + \rangle_a \phi \mid \langle - \rangle_a \phi \mid \langle + \rangle_* \phi \mid \langle - \rangle_* \phi \mid K_i \phi \mid \neg \phi \mid \phi \wedge \phi$$

As usual, we assume the boolean constants $\bot = p \wedge \neg p$ and $\top = \neg \bot$ and the boolean operators $\Rightarrow, \vee, \iff$. In addition we define

$$\begin{array}{ll} [-]_a \phi = \neg \langle - \rangle_a \neg \phi & [+]_a \phi = \neg \langle + \rangle_a \neg \phi, \\ [-]^* \phi = \neg \langle - \rangle^* \neg \phi, & [+]^* \phi = \neg \langle + \rangle^* \neg \phi, \\ \langle - \rangle \phi = \bigvee_{a \in A} \langle - \rangle_a \phi, & \langle + \rangle \phi = \bigvee_{a \in A} \langle + \rangle_a \phi, \\ [-] \phi = \neg \langle - \rangle \neg \phi, & [+] \phi = \neg \langle + \rangle \neg \phi. \end{array}$$

In order to define the semantics we consider the (oriented) labeled graphs over $\mathcal{A}$. These capture sets of histories as we defined them in the previous section.

The nodes of the graph are states and the transitions are labelled by actions in $\mathcal{A}$. A path through the graph is a history.

If $G = (S, \xrightarrow{a})_{a\in\mathcal{A}}$ is a labelled graph, we denote by $\longrightarrow$ the relation $\bigcup_{a\in A} \xrightarrow{a}$ and by $\xrightarrow{*}$ the reflexive-transitive closures of $\longrightarrow$ respectively.

Definition 11 (Labelled forest[5]). *A* labelled forest over $\mathcal{A}$ *is a labelled graph* $G = (S, \xrightarrow{a})_{a\in\mathcal{A}}$ *such that*

1. *for arbitrary* $s, s', s'' \in S$, $s' \longrightarrow s$ *and* $s'' \longrightarrow s$ *implies* $s' = s''$*;*
2. *there exists no infinite sequence* $s_0, s_1, .., s_k, .. \in S$ *such that* $s_{i+1} \longrightarrow s_i$ *for each* $i \in \mathbb{N}$*; i.e. it is well-founded to the past.*

The *support* of a forest $\mathcal{F}$, denoted by $supp(\mathcal{F})$, is the set of its nodes. Give a labelled forest $\mathcal{F}$, we say that an equivalence relation $\approx \subseteq supp(\mathcal{F}) \times supp(\mathcal{F})$ *reflects the branching structure* if whenever $s \approx t$, the existence of a transition $s' \xrightarrow{a} s$ implies the existence of $t' \in supp(\mathcal{F})$ such that $t' \xrightarrow{a} t$ and $s' \approx t'$. Notice that this is a *backward* bisimulation property; it is a backward preservation property.

Definition 12 (Epistemic Frame). *Given a set* $\mathcal{I}$ *(of* agents*),* an epistemic frame *is a tuple* $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i\in\mathcal{I}})$*, where* $\mathcal{F}$ *is a labelled forest over* $\mathcal{A}$ *and* $(\approx_i)_{i\in\mathcal{I}}$ *is an indexed set of equivalence relations on* $supp(\mathcal{F})$ *such that for each* $i \in \mathcal{I}$*,* $\approx_i$ *preserves the branching structure.*

We call the relation $\approx_i$ the *indistinguishability* relation of agent $i \in \mathcal{I}$. Observe that an epistemic frame defines a unique history-LTS and a history-LTS is supported by a unique epistemic frame.

In the following definition we write s, t, r with or without subscripts for states, p and variants for propositions, ϕ, ψ for formulas and a for actions and i for agents.

Definition 13 (Semantics). *The semantics is defined for an epistemic frame* $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i\in\mathcal{I}})$*, a state* $s \in supp(\mathcal{F})$ *and an interpretation function* $Prop : supp(\mathcal{F}) \longrightarrow 2^{\mathcal{P}}$*, as follows.*

$s \models \top$ *for all* s.
$s \models p$ *if* $p \in Prop(s)$.
$s \models \langle + \rangle_a \phi$ *if there exists a state* t *such that* $s \xrightarrow{a} t$ *and* $t \models \phi$.
$s \models \langle - \rangle_a \phi$ *if there exists a state* r *such that* $r \xrightarrow{a} s$ *and* $r \models \phi$.
$s \models \langle + \rangle_* \phi$ *if there exist* $s_1, ..., s_n \in S$ *and* $a_1, ..., a_n \in \mathcal{A}$ *such that* $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} ... \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$ *and* $s_n \models \phi$.
$s \models \langle - \rangle_* \phi$ *if there exist* $s_0, ..., s_{n-1} \in S$ *and* $a_1, ..., a_n \in \mathcal{A}$ *such that* $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s$ *and* $s_0 \models \phi$.
$s \models K_i \phi$ *if for all* t *such that* $s \sim_i t$, $t \models \phi$.
$s \models \neg\phi$ *if it is not the case that* $s \models \phi$.
$s \models \phi_1 \wedge \phi_2$ *if* $s \models \phi_1$ *and* $s \models \phi_2$.

[5] We call it labelled forest because it is a set of labelled trees.

Now we have defined our basic operators. For convenience, we also define other operators as shorthand for certain combinations of these basic operators:

$$\begin{aligned}
\langle + \rangle \phi &:= \bigvee_{a \in \mathcal{A}} \langle + \rangle_a \phi \\
\langle - \rangle \phi &:= \bigvee_{a \in \mathcal{A}} \langle - \rangle_a \phi \\
[+]_a \phi &:= \neg \langle + \rangle_a \neg \phi \\
[-]_a \phi &:= \neg \langle - \rangle_a \neg \phi \\
[+] \phi &:= \bigwedge_{a \in \mathcal{A}} [+]_a \phi \\
[-] \phi &:= \bigwedge_{a \in \mathcal{A}} [-]_a \phi \\
[+]_* \phi &:= \neg \langle + \rangle_* \neg \phi \\
[-]_* \phi &:= \neg \langle - \rangle_* \neg \phi \\
L_i \phi &:= \neg K_i \neg \phi
\end{aligned}$$

Note that $[+]\phi = \neg\langle + \rangle \neg\phi$ and $[-]\phi = \neg\langle - \rangle \neg\phi$. The semantics of these derived operators are:

$s \models \bot$ never.
$s \models [+]_a \phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $s \xrightarrow{a} t$, $t \models \phi$,
$s \models [-]_a \phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $t \xrightarrow{a} s$, $t \models \phi$,
$s \models [+]^* \phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $s \xrightarrow{*} t$, $t \models \phi$,
$s \models [-]^* \phi$ iff for any $t \in supp(\mathcal{F})$ s.t. $t \xrightarrow{*} s$, $t \models \phi$.

If we have an epistemic frame $\mathcal{E}$, a *valuation* is a map $\rho : supp(\mathcal{F}) \to 2^{\mathcal{P}}$ which provides an interpretation of the propositions in the states of $\mathcal{E}$. If a formula ϕ is true in a given epistemic frame $\mathcal{E}$ and state s with a valuation ρ we write $\mathcal{E}, s, \rho \models \phi$ and we say that $(\mathcal{E}, s, \rho)$ is a model of ϕ. In this case we say that ϕ is *satisfiable*. Given an arbitrary $\phi \in \mathcal{L}$, if for any epistemic frame $\mathcal{E} = (\mathcal{F}, (\approx_i)_{i \in \mathcal{I}})$, any state $s \in supp(\mathcal{F})$ and any valuation ρ, $\mathcal{E}, s, \rho \models \phi$ we say that ϕ is *valid* and write $\models \phi$. We also write $\mathcal{E}, s, \rho \models \Phi$, where Φ is a set of formulas if it models every formula in the set Φ. We write $\Gamma \models \phi$ if any model of Γ is a model of ϕ.

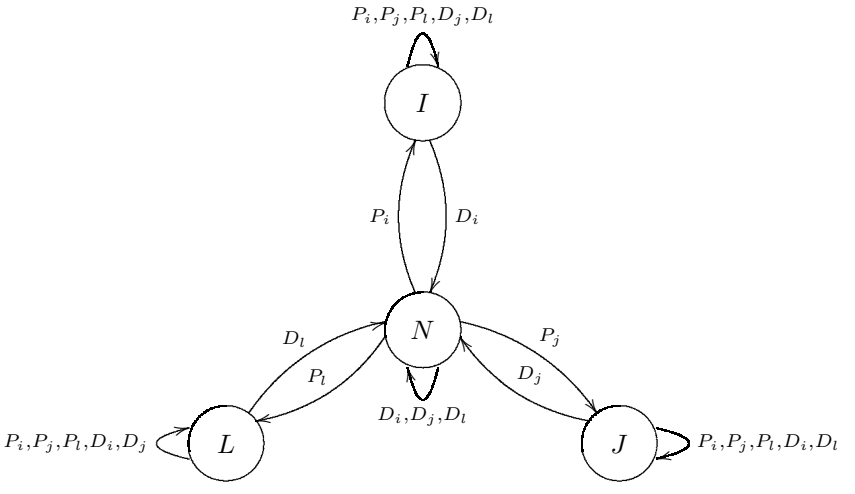
Example 6. Here is a more complicated example with multiple agents[6] which we describe as an illustration of our logic.

The situation is as follows: There are three agents, one diamond, and a bag. The diamond can either be held by one of the agents or it can be in the box. Each agent can perform two actions: reach into the bag and take the diamond *if it is there*, and drop the diamond into the bag, or pretend to drop it. After dropping or pretending to drop the diamond, the agent shows the other agents that his

[6] This example was developed by Caitlin Phillips.

hands are empty, so it is impossible to keep the diamond while pretending to drop it. On the other hand, if the agent does not have the diamond, he can still pretend to drop it in the box. If the agent reaches into the box to take the diamond, he will take it if it is there, and will not take it if it is not there.

Here is the transition system:



The agents are i, j, and l. In state N, no one has the diamond, and in states I, J, and L, agents i, j, and l respectively have the diamond. Action P_i represents agent i picking up or pretending to pick up the diamond and action D_i represents agent i dropping or pretending to drop the diamond.

The equivalence classes are as follows:

$N \sim_i J \sim_i L$
$N \sim_j I \sim_j L$
$N \sim_l I \sim_l J.$

We use as propositions $@I$, $@J$, $@L$ and $@N$; each proposition is true only in the corresponding state and in each state only the corresponding proposition is true. For example, the only proposition true in state I is $@I$. We write Prop for this set of 4 propositions. Now we consider the formulas

$$\phi_1 = \bigwedge_{X \in \mathsf{Prop}} X \Rightarrow K_l X$$

$$\phi_2 = \langle - \rangle_{P_l} @N$$

$$\phi_3 = \bigvee_{X \in \mathsf{Prop}} K_l X$$

The first formula says that if any of the propositions are true then l knows it: in short l knows where the diamond is. Of course this formula is not universally

true, it might or might not be true depending on the situation. The second formula is true for a history where the immediately preceding action is P_l (l picks up the diamond) and in the immediately preceding state nobody had the diamond (i.e. it was in the bag). In other words ϕ_2 describes the situation where the diamond was in the bag and l has just picked it up. The formula ϕ_3 says whatever the state happens to be, l knows it. Here are two formulas that are true in every state of the unfolded labelled transition system (the history LTS):

$$\phi_2 \Rightarrow [+]^*\phi_1 \text{ and } \phi_3 \Rightarrow [+]^*\phi_1.$$

The first is true because l has picked up the diamond and can now track its movements precisely for all future moves since all actions are visible to him. The second statement is slightly more general, it says that once l knows where the diamond is he can track its future exactly.

Here is another example of reasoning within this system. We define ϕ_4 to be like ϕ_1 except that we have K_i instead of K_l and ϕ_5 is like ϕ_1 except that K_j replaces K_l. Now we can conclude that the following formula is true in every state

$$\langle - \rangle_{D_i} \langle - \rangle_{D_j} \langle - \rangle_{D_l} \Rightarrow [+]^*(\phi_1 \wedge \phi_2 \wedge \phi_3).$$

What we cannot say in this logic is that the location of the diamond is common knowledge. □

5 A Complete Axiomatization

We assume the axioms and rules of classical propositional logic. Because we have 5 independent modalities in our logic ($K_i, \langle + \rangle_a, \langle - \rangle_a, \langle + \rangle^*$ and $\langle - \rangle^*$) we expect to have, in addition, five classes of axioms (one for each modality) reflecting the behaviour of that modality in relation to Booleans. In addition, we will have a few other classes of axioms describing the relations between various modalities. For instance, $\langle + \rangle_a$ and $\langle - \rangle_a$ are in a certain duality supported by our intuition about time, so we expect to have some axioms relating these two. Similarly between $\langle + \rangle^*$ and $\langle - \rangle^*$. We also have some clear intuition about the relation between time transition and knowledge update that will be characterized by some axioms combining dynamic and epistemic operators.

The axioms of $\mathcal{L}$ are presented in Table 1.

Many of the lemmas apply generically to $\langle\rangle$ or [] modalities and the proofs are essentially identical for the different variants. To streamline some proofs, we use the tuple of symbols $(\Diamond, \Box)$ to represent an arbitrary tuple of type $(\langle - \rangle_a, [-]_a)$, $(\langle + \rangle_a, [+]_a)$, $(\langle - \rangle, [-])$, or $(\langle + \rangle, [+])$. Similarly, $(\Diamond^*, \Box^*)$ represents $(\langle + \rangle^*, [+]^*)$ or $(\langle - \rangle^*, [-]^*)$. We also use $(\Diamond^x, \Box^x)$ to represent an arbitrary tuple of type $(\langle - \rangle_a, [-]_a)$, $(\langle + \rangle_a, [+]_a)$, $(\langle - \rangle, [-])$, $(\langle + \rangle, [+])$, $(\langle + \rangle^*, [+]^*)$ or $(\langle - \rangle^*, [-]^*)$. With these notations, the axioms (A1),(A2), (B1), (B2), (C1), (C2) and (D1), (D2) cn be regarded as instances of (X1), (X2). Similarly, (C3), (C4) and (D3), (D4) are instances of (X3), (X4).

Table 1. Hilbert-style axiomatization for $\mathcal{L}$

(A1): $\vdash [+]_a\phi \wedge [+]_a(\phi \Rightarrow \psi) \Rightarrow [+]_a\psi$
(A2): If $\vdash \phi$ then $\vdash [+]_a\phi$

(B1): $\vdash [-]_a\phi \wedge [-]_a(\phi \Rightarrow \psi) \Rightarrow [-]_a\psi$
(B2): If $\vdash \phi$ then $\vdash [-]_a\phi$
(B3): $\vdash \langle - \rangle_a \top \Rightarrow \bigwedge_{a \neq b} [-]_b \bot$
(B4): $\vdash \langle - \rangle_a \phi \Rightarrow [-]\phi$

(AB1): $\vdash \phi \Rightarrow [+]_a \langle - \rangle_a \phi$
(AB2): $\vdash \phi \Rightarrow [-]_a \langle + \rangle_a \phi$

(C1): $\vdash [+]^*\phi \wedge [+]^*(\phi \Rightarrow \psi) \Rightarrow [+]^*\psi$
(C2): If $\vdash \phi$ then $\vdash [+]^*\phi$
(C3): $\vdash [+]^*\phi \leftrightarrow (\phi \wedge [+][+]^*\phi)$
(C4): $\vdash [+]^*(\phi \Rightarrow [+]\phi) \Rightarrow (\phi \Rightarrow [+]^*\phi)$
(D1): $\vdash [-]^*\phi \wedge [-]^*(\phi \Rightarrow \psi) \Rightarrow [-]^*\psi$
(D2): If $\vdash \phi$ then $\vdash [-]^*\phi$
(D3): $\vdash [-]^*\phi \leftrightarrow (\phi \wedge [-][-]^*\phi)$
(D4): $\vdash [-]^*(\phi \Rightarrow [-]\phi) \Rightarrow (\phi \Rightarrow [-]^*\phi)$

(BD1): $\vdash \langle - \rangle^* [-] \bot$

(E1): $\vdash K_i\phi \wedge K_i(\phi \Rightarrow \psi) \Rightarrow K_i\psi$
(E2): If $\vdash \phi$ then $\vdash K_i\phi$
(E3): $\vdash K_i\phi \Rightarrow \phi$
(E4): $\vdash K_i\phi \Rightarrow K_iK_i\phi$
(E5): $\vdash \neg K_i\phi \Rightarrow K_i\neg K_i\phi$

(BE1): $\vdash \langle - \rangle_a K_i\phi \Rightarrow K_i \langle - \rangle_a \phi$

(X1): $\vdash \Box^x\phi \wedge \Box^x(\phi \Rightarrow \psi) \Rightarrow \Box^x\psi$
(X2): If $\vdash \phi$ then $\vdash \Box^x\phi$
(X3): $\vdash \Box^*\phi \leftrightarrow (\phi \wedge \Box\Box^*\phi)$
(X4): $\vdash \Box^*(\phi \Rightarrow \Box\phi) \Rightarrow (\phi \Rightarrow \Box^*\phi)$

From (X1) and (X2) alone we can prove a lemma which can be instantiated to all the particular instances. This is a standard lemma of modal logic.

Lemma 1. *1. If* $\vdash \phi \Rightarrow \psi$, *then* $\vdash \Box^x\phi \Rightarrow \Box^x\psi$ *and* $\vdash \Diamond^x\phi \Rightarrow \Diamond^x\psi$.
2. If $\vdash \phi \Rightarrow \psi$, *then* $\vdash K_i\phi \Rightarrow K_i\psi$.
3. $\vdash \langle - \rangle_a\phi \Rightarrow [-]_a\phi$ *and* $\vdash \langle - \rangle\phi \Rightarrow [-]\phi$.

Proof. 1. From (X2), $\vdash \phi \Rightarrow \psi$ implies $\vdash \Box^x(\phi \Rightarrow \psi)$. If we use this with $\vdash \Box^x(\phi \Rightarrow \psi) \Rightarrow (\Box^x\phi \Rightarrow \Box^x\psi)$, which is equivalent to (X1), we obtain $\vdash \Box^x\phi \Rightarrow \Box^x\psi$.

To prove the second implication, we start from $\vdash \neg\psi \Rightarrow \neg\phi$ and apply the first result which gives us $\vdash \Box^x\neg\phi \Rightarrow \Box^x\neg\psi$. Using De Morgan we derive $\vdash \Diamond^x\phi \Rightarrow \Diamond^x\psi$.

2. It is proved in the same way as 1; in fact K is a box-like modality.
3. From (B4) we have $\vdash \langle - \rangle_a \phi \Rightarrow \bigwedge_a [-]_a \phi$ which implies $\vdash \langle - \rangle_a \phi \Rightarrow [-]_a \phi$.

The same axiom implies $\vdash \bigwedge_a (\langle - \rangle_a \phi \Rightarrow [-]\phi)$ which is equivalent to $\vdash \bigvee_a \langle - \rangle_a \phi \Rightarrow [-]\phi$ which implies $\vdash \langle - \rangle \phi \Rightarrow [-]\phi$.

As usual, we say that a formula $\phi \in \mathcal{L}$ is *provable*, denoted by $\vdash \phi$, if it can be proved from the axioms in Table1 and boolean rules. We say that ϕ is *consistent*, if $\neg\phi$ is not provable from the axioms.

Given $\Phi, \Psi \subseteq \mathcal{L}$, Φ proves Ψ if from the formulas of Φ and the axioms we can prove each $\psi \in \Psi$; we write $\Phi \vdash \Psi$. Let $[\Phi] = \{\psi \in \mathcal{L} \mid \Phi \vdash \psi\}$; this is the deductive closure of Φ. Φ is consistent if it is not the case that $\Phi \vdash \bot$.

For a sublanguage $L \subseteq \mathcal{L}$, we call Φ *L-maximally consistent* if Φ is consistent and no formula of L can be added to it without making it inconsistent. The following lemma follows directly from the definition of maximal consistency.

Lemma 2. *If Γ is a consistent set of formulas then the following assertions are true.*

1. *if $\Diamond^x \top \in [\Gamma]$ and $\Diamond^x \phi \notin [\Gamma]$, then $\{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\} \cup \{\neg\phi\}$ is consistent.*
2. *if $\Box^x \phi \notin [\Gamma]$, then $\{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\} \cup \{\neg\phi\}$ is consistent.*

Proof. Let $\Lambda = \{\psi \in \mathcal{L} \mid \Box^x \psi \in [\Gamma]\}$. Suppose that $\Lambda \cup \{\neg\phi\}$ is inconsistent. Then there is a finite set $\{f_1, .., f_n\} \subseteq \Lambda$ s.t. $\vdash f_1 \wedge .. \wedge f_n \Rightarrow \phi$. Hence, $\vdash \Box^x(f_1 \wedge .. \wedge f_n) \Rightarrow \Box^x \phi$ implying further $\vdash (\Box^x f_1 \wedge .. \wedge \Box^x f_n) \Rightarrow \Box^x \phi$. Hence, $\Box^x \phi \in [\Gamma]$.

1. If $\Diamond^x \top \in [\Gamma]$, from $\Box^x \phi \in [\Gamma]$ we obtain $\Diamond^x \phi \in [\Gamma]$ - contradiction.
2. $\Box^x \phi \notin [\Gamma]$ is again contradictory.

A basic theorem that holds for the axiom system is the soundness property.

Theorem 1 (Soundness). *The axiomatic system of $\mathcal{L}$ is sound, i.e., for any $\phi \in \mathcal{L}$,*

$$\vdash \phi \text{ implies } \models \phi.$$

The proof is a routine structural induction. It is sufficient to prove that each axiom is sound and that each rule preserves the soundness.

The more interesting result is the completeness of the axiom system. Moreover, we will show that for each consistent formula a *finite* model can be constructed.

Recall that there are two notions of completeness: *strong* completeness and *weak* completeness. Strong completeness says that

$$\Gamma \models \phi \iff \Gamma \vdash \phi.$$

An important easy consequence of strong completeness is the so-called *compactness* property. A logic is said to be compact if every inconsistent set of formulas has a *finite* inconsistent subset. Our logic is not compact. For example, the set of formulas

$$\{p, [+]p, [+][+]p, [+]^3 p, \ldots, \neg[+]^* p\}$$

is not consistent but any finite subset is consistent. Therefore we cannot hope to prove strong completeness. Instead we prove weak completeness

$$\models \phi \iff \vdash \phi.$$

Many of the basic completeness proofs in the literature are strong completeness proofs and are much easier than weak completeness proofs. The proof that we present shares many of the features of the weak completeness proof for PDL.

Before proceeding with these proofs we establish some notation that will be useful for future constructions.

We extend, canonically, all the logical operators from formulas to sets of formulas. Thus for arbitrary $\Phi, \Psi \subseteq \mathcal{L}$, $\Phi \wedge \Psi = \{\phi \wedge \psi \mid \phi \in \Phi, \psi \in \Psi\}$, $\langle + \rangle_a \Phi = \{\langle + \rangle_a \phi \mid \phi \in \Phi\}$, and so on for all the modal operators.

If $\Phi \subseteq \mathcal{L}$ is finite, we use Φ to also denote $\bigwedge_{\phi \in \Phi} \phi$; it should be clear from the context when Φ denotes a set of formulas and when it denotes the conjunction of its elements.

A key step in the proof is the construction of models by using maximally consistent sets as states. However, because we are trying to prove a weak completeness theorem we have to ensure that we are constructing finite sets of formulas. The liberal notion of maximal consistency used in strong completeness proofs is not available to us. If we wish to construct a model of a formula ϕ, we need to define a special family of formulas associated with ϕ from which we will construct maximal consistent subsets. Furthermore we need to ensure that the collection of formulas we construct is finite. We adapt a construction due to Fischer and Ladner [14] developed in the context of PDL.

For an arbitrary $\phi \in \mathcal{L}$, let $\sim \phi = \psi$ whenever $\phi = \neg\psi$ and $\sim \phi = \neg\phi$ otherwise.

For an arbitrary $\phi \in \mathcal{L}$, let $\overline{k}_i \phi = \phi$ whenever $\phi = K_i \psi$ or $\phi = \neg K_i \psi$ and $\overline{k}_i \phi = K_i \phi$ otherwise.

Definition 14. *The* (Fischer-Ladner) closure of ϕ, *written* $\mathcal{FL}(\phi)$, *is defined as a set of formulas such that:*

- $\phi, \langle - \rangle_a p, \langle - \rangle_a \top \in \mathcal{FL}(\phi)$,
- *if* $\psi \in \mathcal{FL}(\phi)$, *then* $\sim \psi \in \mathcal{FL}(\phi)$, $\overline{k}_i \psi$ *and any subformula of* ψ *is in* $\mathcal{FL}(\phi)$,
- *if* $\langle - \rangle_a \psi \in \mathcal{FL}(\phi)$ *or* $\langle + \rangle_a \psi \in \mathcal{FL}(\phi)$, *then* $\langle - \rangle \psi, \langle + \rangle \psi \in \mathcal{FL}(\phi)$,
- *if* $\Diamond^* \psi \in \mathcal{FL}(\phi)$, *then* $\Diamond \Diamond^* \psi \in \mathcal{FL}(\phi)$.

The following lemma is immediate but important to state because we have to ensure that we always have finite sets of formulas when we construct models out of sets of formulas.

Lemma 3. *For any* $\phi \in \mathcal{L}$, $\mathcal{FL}(\phi)$ *is finite.*

In what follows we fix a consistent formula $\theta \in \mathcal{L}$ and we construct a finite model for θ. This means that we construct an epistemic frame $\mathcal{E}_\theta = (\mathcal{F}_\theta, (\approx_i)_{i \in \mathcal{I}})$, a

valuation $\rho : supp(\mathcal{F}_\theta) \Rightarrow 2^{\mathcal{P}}$ and we will identify a state $s \in supp(\mathcal{F}_\theta)$ such that $s \models \theta$.

Let Ω_θ be the set of $\mathcal{FL}(\theta)$-maximally consistent sets. Because $\mathcal{FL}(\theta)$ is finite, Ω_θ and any $\Gamma \in \Omega_\theta$ are finite sets. In the construction of the model we will use Ω_θ as the support set for $\mathcal{F}_\theta$. The transitions on Ω_θ are defined as follows. For each $a \in \mathcal{A}$, let $\xrightarrow{a} \subseteq \Omega_\theta \times \Omega_\theta$ be defined by

$$\Gamma \xrightarrow{a} \Gamma' \text{ iff for any } \psi \in \mathcal{L}, [+]_a\psi \in [\Gamma] \text{ implies } \psi \in [\Gamma'].$$

Now we prove a few properties of these transitions that will be important for the rest of the proof.

Lemma 4. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$ the following are equivalent*

1. *for any $\phi \in \mathcal{L}$, $[+]_a\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$,*
2. *for any $\phi \in \mathcal{L}$, $[-]_a\phi \in [\Gamma']$ implies $\phi \in [\Gamma]$.*

Proof. (1) implies (2): Suppose that $[-]_a\phi \in [\Gamma']$. Then, $\vdash \Gamma' \Rightarrow [-]_a\phi$ and using axiom (AB1), $\vdash \langle+\rangle_a\Gamma' \Rightarrow \phi$. If we prove that $\langle+\rangle_a\Gamma' \in [\Gamma]$, then $\phi \in [\Gamma]$ and the proof is done. Observe that $\langle+\rangle_a\top \in [\Gamma]$ because otherwise $\neg\langle+\rangle_a\top \in [\Gamma]$ implying $[+]_a\bot \in [\Gamma]$ and from the hypothesis we obtain $\bot \in [\Gamma']$ - impossible. Hence, $\langle+\rangle_a\top \in [\Gamma]$ and if $\langle+\rangle_a\Gamma' \notin [\Gamma]$, from Lemma 2 instantiated to $\Box^x = [+]_a$, we obtain that $\{\psi \mid [+]_a\psi \in [\Gamma]\} \cup \{\neg\Gamma'\}$ is consistent. But this is impossible because, from the hypothesis, $\{\psi \mid [+]_a\psi \in [\Gamma]\} \subseteq [\Gamma']$.

(2) implies (1) Suppose that $[+]_a\phi \in [\Gamma]$. Then, $\vdash \Gamma \Rightarrow [+]_a\phi$ implying $\vdash \langle-\rangle_a\Gamma \Rightarrow \langle-\rangle_a[+]_a\phi$. Now (AB2) guarantees that $\vdash \langle-\rangle_a\Gamma \Rightarrow \phi$. In any normal modal logic we have that $\vdash (\Box\psi \wedge \Diamond\top) \Rightarrow \Diamond\psi$. We use this with the previous formula and we obtain $\vdash ([-]_a\Gamma \wedge \langle-\rangle_a\top) \Rightarrow \phi$.

Note that $\langle-\rangle_a\top \in \Gamma'$ because otherwise $[-]_a\bot \in \Gamma'$ and, from the hypothesis we obtain that $\bot \in [\Gamma]$ - impossible. Now, if we prove that $[-]_a\Gamma \in [\Gamma']$, then $\phi \in [\Gamma']$ and the proof is done. Now note that $[-]_a\Gamma \notin [\Gamma']$ implies, using Lemma 2 instantiated with $\Box^x = [-]_a$, that $\{\psi \mid [-]_a\psi \in [\Gamma']\} \cup \{\neg\Gamma\}$ is consistent. But this is impossible because, from the hypothesis, $\{\psi \mid [-]_a\psi \in [\Gamma']\} \subseteq [\Gamma]$.

This lemma tells us that we can define the transitions either using $[+]$ or $[-]$.

Lemma 5. *For arbitrary $\Gamma \in \Omega_\theta$ and $[+]_a\phi \in \mathcal{FL}(\theta)$,*

1. *$[+]_a\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$, $\Gamma \xrightarrow{a} \Gamma' \Rightarrow \phi \in \Gamma'$;*
2. *$\langle+\rangle_a\phi \in \Gamma$ iff there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \xrightarrow{a} \Gamma', \phi \in \Gamma'$;*
3. *$[-]_a\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma, \phi \in \Gamma'$;*
4. *$\langle-\rangle_a\phi \in \Gamma$ iff there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma, \phi \in \Gamma'$.*

Proof. 1. ($\Rightarrow$:) From the definition of $\xrightarrow{a}$.
($\Leftarrow$:) Let ϕ be such that $\phi \in [\Gamma']$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \xrightarrow{a} \Gamma'$. We need to prove that $[+]_a\phi \in [\Gamma]$. Note that a formula that is in $[\Gamma]$ and also in $\mathcal{FL}(\theta)$ is automatically in Γ.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \xrightarrow{a} \Gamma'\}$ and let $\delta = \bigvee_{\Gamma' \in \Delta} \Gamma'$. Obviously, $\vdash \delta \Rightarrow \phi$ implying $\vdash [+]_a\delta \Rightarrow [+]_a\phi$. Now, if we prove that $[+]_a\delta \in [\Gamma]$, the proof is done.

Suppose that $[+]_a\delta \notin [\Gamma]$. Lemma 2 implies that $\Lambda \cup \{\neg\delta\}$ is consistent, where $\Lambda = \{\psi \mid [+]_a\psi \in [\Gamma]\}$. But $[+]_a\psi \in [\Gamma]$ implies $\psi \in \Gamma'$ for each $\Gamma' \in \Delta$ and this proves that $\Lambda \cup \{\neg\delta\}$ cannot be consistent.

(2) is the De Morgan dual of (1).

(3) and (4) are proved in the same way as (1) and (2).

We draw the reader's attention to a minor subtlety in the proof because it recurs in several later proofs. We showed that a formula in $\mathcal{FL}(\theta)$, say ϕ, is in the *deductive closure* of a maximally consistent subset, say Γ, of $\mathcal{FL}(\theta)$, in other words we showed that $\phi \in [\Gamma]$. From the fact that ϕ is itself in $\mathcal{FL}(\theta)$ we were able to deduce that ϕ is in Γ itself precisely because Γ is maximal consistent as a subset of $\mathcal{FL}(\theta)$.

We now need to establish the analogous results for the starred modalities. In what follows, let $\rightarrow = \bigcup_{a\in\mathcal{A}} \xrightarrow{a}$ and $\rightarrow^*$ be its reflexive-transitive closure. This means that $\Gamma \rightarrow^* \Gamma'$ if there exists a sequence $\Gamma_1, \ldots, \Gamma_k \in \Omega_\theta$ such that

$$\Gamma = \Gamma_1 \rightarrow \Gamma_2 \rightarrow \ldots \rightarrow \Gamma_{k-1} \rightarrow \Gamma_k = \Gamma';$$

Because $\rightarrow^*$ is reflexive, k can be 1.

Lemma 6. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$ the following are equivalent*

1. *for any $\phi \in \mathcal{L}$, $[+]^*\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$,*
2. *for any $\phi \in \mathcal{L}$, $[-]^*\phi \in [\Gamma']$ implies $\phi \in [\Gamma]$,*
3. *$\Gamma \rightarrow^* \Gamma'$.*

Proof. (1) $\Longrightarrow$ (3): Let $\Delta = \{\Lambda \in \Omega_\theta \mid \Gamma \rightarrow^* \Lambda\}$ and $\delta = \bigvee_{\Lambda\in\Delta} \Lambda$.

By construction, if $[+]\phi \in [\Lambda]$ for some $\Lambda \in \Delta$, there exists $\Lambda' \in \Delta$ such that $\phi \in [\Lambda']$. This entails $\vdash \delta \Rightarrow [+]\delta$ which guarantees that $\vdash [+]^*(\delta \Rightarrow [+]\delta)$. Using axiom (C4), we obtain $\vdash \delta \Rightarrow [+]^*\delta$. But $\Gamma \in \delta$ (because $\rightarrow^*$ is reflexive), consequently $\vdash \Gamma \Rightarrow \delta$. From here and the previous we derive $\vdash \Gamma \Rightarrow [+]^*\delta$ implying $[+]^*\delta \in [\Gamma]$. Now using 1., $\delta \in [\Gamma']$ implying $\Gamma' \in \Delta$.

(3) $\Rightarrow$ (1): Suppose that $\Gamma = \Gamma_1 \rightarrow \ldots \rightarrow \Gamma_k = \Gamma'$ and $[+]^*\phi \in [\Gamma]$. Axiom (C3) guarantees that $\phi \in [\Gamma_1]$ and $[+][+]^*\phi \in [\Gamma_1]$. Hence $[+]^*\phi \in [\Gamma_2]$ from the definition of $\rightarrow$. The same argument can be repeated for the k cases eventually giving $[+]^*\phi \in [\Gamma_k] = [\Gamma']$ which implies, using axiom (C3), $\phi \in [\Gamma']$.

(2) $\Leftrightarrow$ (3): It is proved in the same way using the axioms (D1) and (D2) in instances of Lemma 1 and (D3), (D4) respectively.

Lemma 7. *For arbitrary $\Gamma \in \Omega_\theta$ and $[+]^*\phi \in \mathcal{FL}(\theta)$,*

1. *$[+]^*\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma', \phi \in \Gamma'$;*
2. *$\langle+\rangle^*\phi \in \Gamma$ iff there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma', \phi \in \Gamma'$;*
3. *$[-]^*\phi \in \Gamma$ iff for any $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma, \phi \in \Gamma'$;*
4. *$\langle-\rangle^*\phi \in \Gamma$ iff there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma, \phi \in \Gamma'$.*

Proof. (1) $\Rightarrow$: From Lemma 6.
($\Leftarrow$:) Let ϕ be such that $\phi \in [\Gamma']$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \longrightarrow^* \Gamma'$. We need to prove that $[+]^*\phi \in [\Gamma]$.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \longrightarrow^* \Gamma'\}$ and let $\delta = \bigvee_{\Gamma' \in \Delta} \Gamma'$. Obviously, $\vdash \delta \Rightarrow \phi$ implying $\vdash [+]^*\delta \Rightarrow [+]^*\phi$. Now, if we prove that $[+]^*\delta \in [\Gamma]$, the proof is done.

Suppose that $[+]^*\delta \notin [\Gamma]$. Lemma 2 implies that $\Lambda \cup \{\neg\delta\}$ is consistent, where $\Lambda = \{\psi \mid [+]^*\psi \in [\Gamma]\}$. But $[+]^*\psi \in [\Gamma]$ implies $\psi \in \Gamma'$ for each $\Gamma' \in \Delta$ and this proves that $\Lambda \cup \{\neg\delta\}$ cannot be consistent.

(2) is equivalent to (1).

(3) and (4) are proved in the same way.

Now we can proceed with our construction of the model for θ. We start by showing that $(\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ is a forest. For this we need to verify that the past is unique and that the graphs have no loops. The precise statement is given in the following theorem.

Theorem 2. *If $f \in \mathcal{L}$ is consistent, then $\mathcal{F}_\theta = (\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ is a forest over $\mathcal{A}$.*

The proof of this theorem is broken down into two lemmas.

Lemma 8. *For arbitrary $\Gamma, \Gamma_1, \Gamma_2 \in \Omega_\theta$, if $\Gamma_1 \xrightarrow{a} \Gamma$ and $\Gamma_2 \xrightarrow{b} \Gamma$, then $a = b$ and $\Gamma_1 = \Gamma_2$.*

Proof. To prove that $a = b$ it is sufficient to observe that $\langle - \rangle_a \top \wedge \langle - \rangle_b \top$ is inconsistent, result that is a direct consequence of axiom (B3).

Now, from $\Gamma_1 \xrightarrow{a} \Gamma$ and $\Gamma_2 \xrightarrow{a} \Gamma$ we prove that $\Gamma_1 = \Gamma_2$. Suppose that there exists $\phi \in \mathcal{FL}(\theta)$ s.t. $\phi \in \Gamma_1$ and $\neg\phi \in \Gamma_2$. Then, from axiom (AB1) we obtain that $[+]_a\langle - \rangle_a\phi \in [\Gamma_1]$ and $[+]_a\langle - \rangle_a\neg\phi \in [\Gamma_2]$. Now $\Gamma_1 \xrightarrow{a} \Gamma$ guarantees that $\langle - \rangle_a\phi \in [\Gamma]$ while $\Gamma_2 \xrightarrow{a} \Gamma$ guarantees that $\langle - \rangle_a\neg\phi \in [\Gamma]$. Further, using axiom (B4) we obtain that $[-]\phi, [-]\neg\phi \in [\Gamma]$ implying $[-]\bot \in [\Gamma]$. On the other hand, $\langle - \rangle_a\phi \in [\Gamma]$ implies $\langle - \rangle_a\top \in [\Gamma]$ which is equivalent to $\neg[-]\bot \in [\Gamma]$ - contradicts the consistency of $[\Gamma]$.

Now we prove that in the graph $(\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ there are no backwards infinite sequences; this will conclude the proof that $(\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ is a forest over $\mathcal{A}$.

Lemma 9. *There exists no infinite sequence $\Gamma_1, \ldots, \Gamma_k, .. \in \Omega_\theta$ such that*

$$..\Gamma_k \longrightarrow \Gamma_{k-1} \longrightarrow .. \longrightarrow \Gamma_1 = \Gamma.$$

Proof. Suppose that there exists such a sequence. Axiom (BD1) guarantees that $\langle - \rangle^*[-]\bot \in [\Gamma]$ and using Lemma 7 we obtain that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \longrightarrow^* \Gamma$ and $[-]\bot \in \Gamma'$. Lemma 8 guarantees that Γ' is one of the elements of our sequence, hence $\neg\langle - \rangle\top \in \Gamma'$. But this implies that there exists no $\Gamma'' \in \Omega_\theta$ such that $\Gamma'' \longrightarrow^* \Gamma'$, this contradiction establishes the result.

To complete the construction of the model for θ we need to define the indistinguishability relations on Ω_θ that will eventually organize our forest as an epistemic frame.

For each $i \in \mathcal{I}$, let $\approx_i \subseteq \Omega_\theta \times \Omega_\theta$ be defined as follows:

$$\Gamma \approx_i \Gamma' \text{ iff for any } \phi \in \mathcal{L}, K_i\phi \in [\Gamma] \text{ iff } K_i\phi \in [\Gamma'].$$

By construction, $\approx_i$ is an equivalence relation. Now, to finalize our construction, we must prove that for each $i \in \mathcal{I}$, $\approx_i$ preserves the branching structure of $\mathcal{F}_\theta$ and finally that we have an epistemic frame.

Theorem 3. *$\mathcal{E}_\theta = (\mathcal{F}_\theta, (\approx_i)_{i\in\mathcal{I}})$, where $\mathcal{F}_\theta = (\Omega_\theta, \xrightarrow{a})_{a\in\mathcal{A}}$ and $\approx_i$ are defined as before, is an epistemic frame.*

The proof is broken into a number of lemmas. The first lemma that we need is the following.

Lemma 10. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$, if for any ϕ, $K_i\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$, then for any ϕ, $K_i\phi \in [\Gamma]$ implies $K_i\phi \in [\Gamma']$.*

Proof. Suppose that for any ϕ, $K_i\phi \in [\Gamma]$ implies $\phi \in [\Gamma']$ and let $K_i\psi \in [\Gamma]$. From our hypothesis we obtain that if $K_i\psi \notin [\Gamma']$, then $K_iK_i\psi \notin [\Gamma]$. From the axioms (E3) and (E4), $\vdash K_i\psi \leftrightarrow K_iK_i\psi$. Hence, $K_i\psi \notin [\Gamma]$, this contradiction completes the proof.

Now we can prove that for each $i \in \mathcal{I}$, $\approx_i$ preserves the backwards branching structure of $\mathcal{F}_\theta$.

Theorem 4. *For arbitrary $\Gamma, \Gamma' \in \Omega_\theta$, if $\Gamma \approx_i \Gamma'$ and there exists $\Gamma_0 \in \Omega_\theta$ such that $\Gamma_0 \xrightarrow{a} \Gamma$, then there exists $\Gamma_0' \in \Omega_\theta$ such that $\Gamma_0' \xrightarrow{a} \Gamma'$ and $\Gamma_0' \approx_i \Gamma_0$.*

Proof. Because $\vdash \top$, using (E2) we obtain $\vdash K_i\top$. Because $K_i\top \in [\Gamma_0]$, we obtain that $\langle -\rangle_a K_i\top \in [\Gamma]$ and axiom (BE1) implies $K_i\langle -\rangle_a\top \in [\Gamma]$. Now, from $\Gamma \approx_i \Gamma'$, $\langle -\rangle_a\top \in [\Gamma']$. From Lemma 5 we obtain that there exists $\Gamma_0' \in \Omega_\theta$ such that $\Gamma_0' \to \Gamma'$.

We prove now that $\Gamma_0' \approx \Gamma_0$. Suppose that $K_i\phi \in [\Gamma_0]$. Then, $\langle -\rangle_a K_i\phi \in [\Gamma]$ and axiom (BE1) implies $K_i\langle -\rangle_a\phi \in [\Gamma]$. Now from $\Gamma \approx_i \Gamma'$, $\langle -\rangle_a\phi \in [\Gamma']$. Now axiom (B4) implies $[-]\phi \in [\Gamma']$ and because $\Gamma_0' \to \Gamma'$, Lemma 5 implies $\phi \in [\Gamma_0']$.

Hence, $K_i\phi \in [\Gamma_0]$ implies $\phi \in [\Gamma_0']$ and Lemma 10 concludes that $K_i\phi \in [\Gamma_0]$ implies $K_i\phi \in [\Gamma_0']$. Similarly can be proved that $K_i\phi \in [\Gamma_0']$ implies $K_i\phi \in [\Gamma_0]$.

Lemma 10 also establishes the next result that is needed for the proof of the theorem.

Lemma 11. *For arbitrary $\Gamma \in \Omega_\theta$ and $K_i\phi \in \mathcal{FL}(\theta)$,*

$$K_i\phi \in \Gamma \text{ iff for any } \Gamma' \in \Omega_\theta \text{ such that } \Gamma \approx_i \Gamma', \phi \in \Gamma'$$

Proof. ($\Rightarrow$) This follows directly from Lemma 10.

($\Leftarrow$) Let ϕ be such that $K_i\phi \in \mathcal{FL}(\theta)$ and $\phi \in \Gamma'$ for each $\Gamma' \in \Omega_\theta$ with $\Gamma \approx_i \Gamma'$. We need to prove that $K_i\phi \in \Gamma$.

Let $\Delta = \{\Gamma' \in \Omega_\theta \mid \Gamma \approx_i \Gamma'\}$, let $\Lambda = \{f_1, \ldots, f_n\} = \bigcap_{\Gamma'\in\Delta} \Gamma'$ and let $F = f_1 \wedge \ldots \wedge f_n$. Then $\vdash F \Rightarrow \phi$ implying $\vdash K_iF \Rightarrow K_i\phi$. Consequently, if we prove that $K_iF \in [\Gamma]$, the proof is done.

Suppose that $K_iF \notin [\Gamma]$. Then, there exists $f_t \in \Lambda$ such that $K_if_t \notin \Gamma$. Then, $\neg K_if_t \in \Gamma$ and axiom (E5) implies $K_i\neg K_if_t \in [\Gamma]$. The definition of $\approx_i$ guarantees that for any $\Gamma' \in \Delta$, $K_i\neg K_if_t \in [\Gamma']$ and axiom (E3) entails that for any $\Gamma' \in \Delta$, $\neg K_if_t \in \Gamma'$. Hence, $\vdash F \Rightarrow \neg K_if_t$ which is equivalent to $\vdash K_if_t \Rightarrow \neg F$. But $\vdash F \Rightarrow f_t$ implying $\vdash K_iF \Rightarrow K_if_t$. Consequently, $\vdash K_iF \Rightarrow \neg F$. But from axiom (E3), $\vdash K_iF \Rightarrow F$, implying $\vdash \neg K_iF$. But Λ is consistent and $K_iF \notin [\Lambda]$, then a similar argument with the one used in Lemma 2 (notice that K_i is a normal modal operator of type $\Box$) shows that $\Lambda \cup \{\neg F\}$ is consistent, which is impossible.

This completes the proof of the theorem.

We are now ready to complete the construction of the model of θ. $\mathcal{E}_\theta$ is the epistemic frame of the model and the we define a valuation $\rho_\theta : \Omega_\theta \longrightarrow 2^{\mathcal{P}}$ by $\rho_\theta(\Gamma) = \{p \in \mathcal{P} \mid p \in \Gamma\}$. With this definition we prove the Truth Lemma.

Lemma 12 (Truth Lemma). *If $\theta \in \mathcal{L}$ is consistent, $\mathcal{E}_\theta$ and ρ_θ are defined as before, then for any $\phi \in \mathcal{FL}(\theta)$ and $\Gamma \in \Omega_\theta$,*

$$\phi \in \Gamma \text{ iff } \Gamma \models \phi.$$

Proof. Induction on ϕ.

[The case $\phi = p \in \mathcal{P}$:] from definition of $Prop_\theta$.

[The case $\phi = \neg\psi$:] ($\Longrightarrow$) Suppose that $\Gamma \not\models \neg\psi$. Then $\Gamma \models \psi$ and from the inductive hypothesis, $\psi \in \Gamma$, hence $\phi \notin \Gamma$.

($\Longleftarrow$) Suppose that $\Gamma \models \neg\psi$ and $\neg\psi \notin \Gamma$. Then, $\psi \in \Gamma$ and the inductive hypothesis guarantees that $\Gamma \models \psi$ - contradiction.

[The case $\phi = \phi_1 \wedge \phi_2$:] $\phi_1 \wedge \phi_2 \in \Gamma$ iff $\phi_1, \phi_2 \in \Gamma$ which is equivalent, using the inductive hypothesis, to [$\Gamma \models \phi_1$ and $\Gamma \models \phi_2$], equivalent to $\Gamma \models \phi_1 \wedge \phi_2$.

[The case $\phi = \langle+\rangle_a\psi$:] ($\Longrightarrow$) If $\langle+\rangle_a\psi \in \Gamma$, Lemma 5 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \xrightarrow{a} \Gamma'$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.

($\Longleftarrow$) $\Gamma \models \langle+\rangle_a\psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \xrightarrow{a} \Gamma'$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 5 implies $\langle+\rangle_a\psi \in \Gamma$.

[The case $\phi = \langle-\rangle_a\psi$:] ($\Longrightarrow$) If $\langle-\rangle_a\psi \in \Gamma$, Lemma 5 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.

($\Longleftarrow$) $\Gamma \models \langle-\rangle_a\psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \xrightarrow{a} \Gamma$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 5 implies $\langle-\rangle_a\psi \in \Gamma$.

[The case $\phi = \langle+\rangle^*\psi$:] ($\Longrightarrow$) If $\langle+\rangle^*\psi \in \Gamma$, Lemma 7 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma'$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.

($\Longleftarrow$) $\Gamma \models \langle+\rangle^*\psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma \rightarrow^* \Gamma'$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 7 implies $\langle+\rangle^*\psi \in \Gamma$.

[The case $\phi = \langle-\rangle^*\psi$:] ($\Longrightarrow$) If $\langle-\rangle^*\psi \in \Gamma$, Lemma 7 implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma$ and $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.

($\Longleftarrow$) $\Gamma \models \langle - \rangle^* \psi$ implies that there exists $\Gamma' \in \Omega_\theta$ such that $\Gamma' \rightarrow^* \Gamma$ and $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 7 implies $\langle - \rangle^* \psi \in \Gamma$.

[The case $\phi = K_i\psi$:] ($\Longrightarrow$) If $K_i\psi \in \Gamma$, Lemma 11 implies that for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \approx_i \Gamma'$, $\psi \in \Gamma'$. From the inductive hypothesis, $\Gamma' \models \psi$, implying $\Gamma \models \phi$.

($\Longleftarrow$) $\Gamma \models K_i\psi$ implies that for any $\Gamma' \in \Omega_\theta$ such that $\Gamma \approx_i \Gamma'$, $\Gamma' \models \psi$. From the inductive hypothesis, $\psi \in \Gamma'$ and Lemma 11 implies $K_i\psi \in \Gamma$.

A direct consequence of Truth Lemma is the finite model property.

Theorem 5 (Finite Model Property). *For any consistent formula $\phi \in \mathcal{L}$ there exists a finite model. Moreover, the size of the model is bound by the structure of ϕ.*

The finite model property in this context has two important consequences: the weak completeness of the axiomatic system and the decidability of the satisfiability problem.

Theorem 6 (Weak Completeness). *The axiomatic system of $\mathcal{L}$ is complete, i.e., for any $\phi \in \mathcal{L}$,*

$$\models \phi \text{ implies } \vdash \phi.$$

Proof. The proof is based on the fact that any consistent formula has a model. We wish to show that $\models \phi$ implies $\vdash \phi$. Now we have shown that if ϕ is consistent it has a model. Clearly then, if $\neg\phi$ is consistent there is a model of $\neg\phi$. The last statement is equivalent to saying that if $\nvdash \phi$ then $\neg\phi$ is satisfiable. If $\neg\phi$ is satisfiable it follows that not every model models ϕ, i.e. $\not\models \phi$. Thus we have $\nvdash \phi$ implies $\not\models \phi$, or taking the contrapositive, $\models \phi$ implies $\vdash \phi$.

Observe that in the previous construction, the size of Ω_θ depends on the number and type of operators that θ contains. In what follows we refer to the cardinality $|\Omega_\theta|$ of Ω_θ as *the size of* θ.

The satisfiability problem is the problem of deciding, given an arbitrary formula $\phi \in \mathcal{L}$, if ϕ has at least one model. The finite model property entails that the satisfiability problem for our logic is decidable.

Theorem 7 (Decidability). *The satisfiability problem for $\mathcal{L}$ is decidable.*

Proof. We have proved that θ has at least one model iff it is consistent. And if θ is consistent we have proved that it has a model of size $|\Omega_\theta| \in \mathbb{N}$. But the class of models of size $k \in \mathbb{N}$ is finite. Consequently, we can decide in a finite number of steps if θ does or does not have a model by checking all the models of the appropriate sizes.

6 Conclusions and Related Work

There seems to be a mysterious divide between concurrency theory, which is primarily a European enterprise, and distributed systems theory which is intensively explored in the United States, Israel and a few other places. This is

unfortunate because the two have much to learn from each other. Concurrency theorists can learn sophisticated new tools like algebraic topology and deeper problems whereas the distributed systems community could learn about, for example, compositional reasoning. Epistemic logic is one of the areas where the distributed systems community got an early start [5] in the mid 1980s whereas the concurrency theory community is only just starting to use these ideas. This schizophrenia is manifested even in the work of individuals! For example, the third author of the present paper worked on common knowledge in asynchronous distributed systems in the late 1980s [15,16] and later on concurrency theory [17] without making the connection. The present work is intended to make epistemic logic more readily accessible to the concurrency theory community by providing a combination of epistemic logic with the Hennessy-Milner logic that the concurrency community is accustomed to using.

The ground breaking paper of Halpern and Moses [2,12] showed the importance of common knowledge as a way of formalizing agreement protocols in distributed systems. Very quickly variants of common knowledge were developed [18,15] and many new applications were explored [19]. Extensions to probability [20] and zero-knowledge protocols [21] quickly followed. The textbook of Fagin et al. [5] made these ideas widely accessible and stimulated even more interest and activity. There are numerous recent papers by Halpern and his collaborators, Parikh and his collaborators and students, van Benthem and the Amsterdam school and by several other authors as well. Applications of epistemic concepts range across game theory, economics, spatial reasoning and even social systems.

In the concurrency theory community there is very little work on this topic. Two striking examples are a recent paper by Chadha, Delaune and Kremer [3] and one by Dechesne, Mousavi and Orzan [22]. The former paper defines an epistemic logic for a particular process calculus, a variant of the π-calculus and uses it to reason about epistemic situations. The latter paper explores the connection between operational semantics and epistemic logic and is closer in spirit to our work which is couched in terms of labelled transition systems. Neither of these paper really integrate Hennessy-Milner logic and epistemic logic. In [23,24], Mardare proposes a complete logic for CCS which combines Hennessy-Milner, epistemic and spatial operators. A recent paper by Knight et al. [25] uses a rudimentary epistemic logic to capture epistemic strategies for games on concurrent processes. A recent paper by Pacuit and Simon [26] develops a PDL-style logic for reasoning about protocols. They also prove a completeness theorem for their logic; it is perhaps the closest in spirit to our work.

Acknowledgments. We have benefitted from helpful discussions on epistemic logic with Alexandru Baltag, Kostas Chatzikokolakis, Valentin Goranko, Joe Halpern, Eric Pacuit, Rohit Parikh, Caitlin Phillips, Doina Precup, R. Ramanujam and Mehrnoosh Sadrzadeh. This research was supported by a grant from NSERC and from an EPSRC grant that allowed Prakash Panangaden to spend a sabbatical year at Oxford. Radu Mardare would like to acknowledge the support of Sapere Aude: DFF-Young Researchers Grant 10-085054 of the Danish

Council for Independent Research. We are very grateful to Alexandra Silva and Robert Constable for the invitation to submit this article to Dexter Kozen's Festschrift.

References

1. Kozen, D., Parikh, R.: An elementary proof of the completeness of PDL. Theoretical Computer Science 14, 113–118 (1981)
2. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. In: Proceedings of the Third ACM Symposium on Principles of Distributed Computing, pp. 50–61 (1984); A revised version appears as IBM Research Report RJ 4421 (August 1987)
3. Chadha, R., Delaune, S., Kremer, S.: Epistemic Logic for the Applied Pi Calculus. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS/FORTE 2009. LNCS, vol. 5522, pp. 182–197. Springer, Heidelberg (2009)
4. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Synthese Library, vol. 337. Springer, Heidelberg (2008)
5. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press (1995)
6. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001)
7. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
8. Milner, R.: Operational and Algebraic Semantics of Concurrent Processes. In: Handbook of Theoretical Computer Science, vol. B, pp. 1201–1242. MIT Press (1990)
9. Popkorn, S.: First Steps in Modal Logic. Cambridge University Press (1994)
10. Kripke, S.: Semantical analysis of modal logic. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9, 67–96 (1963)
11. Hintikka, J.: Knowledge and Belief. Cornell University Press (1962)
12. Halpern, J., Moses, Y.: Knowledge and common knowledge in a distributed environment. JACM 37, 549–587 (1990)
13. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32, 137–162 (1985)
14. Fischer, M., Ladner, R.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences (1979)
15. Panangaden, P., Taylor, K.E.: Concurrent common knowledge. In: Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, pp. 197–209 (1988)
16. Panangaden, P., Taylor, K.E.: Concurrent common knowledge. Distributed Computing 6, 73–93 (1992)
17. Saraswat, V.A., Rinard, M., Panangaden, P.: Semantic foundations of concurrent constraint programming. In: Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages (1991)
18. Neiger, G., Toueg, S.: Substituting for real time and common knowledge in asynchronous distributed systems. In: Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing, pp. 281–293 (1987)
19. Neiger, G., Tuttle, M.R.: Common Knowledge and Consistent Simultaneous Coordination. In: van Leeuwen, J., Santoro, N. (eds.) WDAG 1990. LNCS, vol. 486, pp. 334–352. Springer, Heidelberg (1991)

20. Halpern, J.Y., Tuttle, M.R.: Knowledge, probability and adversaries. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 103–118. ACM (1989)
21. Halpern, J.Y., Moses, Y., Tuttle, M.R.: A knowledge-based analysis of zero knowledge. In: Proceedings of the 20th ACM Symposium on Theory of Computing, pp. 132–147 (1988)
22. Dechesne, F., Mousavi, M.R., Orzan, S.: Operational and Epistemic Approaches to Protocol Analysis: Bridging the Gap. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 226–241. Springer, Heidelberg (2007)
23. Mardare, R., Priami, C.: A decidable extension of Hennessy-Milner logic with spatial operators. Technical Report DIT-06-009, Ingegneria e Scienza dell'Informazione, University of Trento (2006)
24. Mardare, R., Priami, C.: Decidable Extensions of Hennessy-Milner Logic. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 196–211. Springer, Heidelberg (2006)
25. Chatzikokolakis, K., Knight, S., Panangaden, P., Palamidessi, C.: Epistemic strategies and games on concurrent processes. ACM Transactions on Computational Logic (in press, 2012)
26. Pacuit, E., Simon, S.: Reasoning with protocols under imperfect information. The Review of Symbolic Logic 4, 412–444 (2011)

Choice and Uncertainty in Games

Rohit Parikh[1,2], Çağıl Taşdemir[2], and Andreas Witzel[3]

[1] Brooklyn College of the City University of New York
rparikh@gc.cuny.edu
[2] Graduate Center of the City University of New York
[3] Courant Institute, New York University

Abstract. We consider an agent choosing between two acts A, B, whose outcomes are uncertain and depend on factors which the agent does not fully know. But for each pair of possible outcomes the agent does know how she would choose. Does the agent then have a way of choosing between the acts which will work at least some of the time?

Dedicated to Dexter Kozen on the Occasion of His Sixtieth Birthday

1 Introduction

Sometimes, we have to choose between two or more actions, but lacking information, we are not sure what outcome will come about as a result of our choice. How should we choose? How might we choose?

To see this a bit formally, suppose an agent is in a situation of uncertainty where she has to choose between two moves L and R but does not know for sure what the outcome will be with either choice. Assume moreover that the agent has no way of assigning probabilities to the various outcomes. In the absence of such information, how might the agent choose?

One option is the maxmin route. An agent can choose L if the *worst* possible outcome with L is better than the *worst* outcome with R. We will describe such an agent as *conservative*. However, an ambitious agent may choose R if the *best* outcome under R is better than the *best* outcome with L. We will describe such an agent as *aggressive*.[1]

It is clear then that in the same situation, an aggressive agent with the same options and the same preferences as a conservative one may still make a different choice.

Some people never buy lottery tickets on the ground that the worst outcome under buying, namely losing one's money, is worse than the certain outcome (no gain, no loss) under not buying. But those who do buy such tickets are clearly judging by the best outcome.

[1] If we did have cardinal utilities and probabilities we would have used the expressions *risk averse* and *risk loving*.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 244–255, 2012.

Such points of view are often taken into account by stockbrokers advising people on investments. A younger investor may prefer a stock with a high potential growth but significant risk. An investor close to retirement age may, on the contrary prefer a stock with less growth but also less risk. A middle aged investor may accept a moderate amount of risk.

In most of this paper we assume that utilities are ordinal. In other words, between any two choices a and b, the agent may be neutral, prefer a or prefer b. Numbers $u(a), u(b)$ can be assigned to a and b so that $u(a) < u(b)$ iff b is preferred to a. However, ordinal utilities are preserved by all order preserving transformations. If c is preferred to b and b to a (which we may write $c > b > a$) then there is no difference between utility assignments to a, b, c of 1, 2, 10 or 1, 9, 10. It is also generally assumed that comparing utilities between different players makes no sense.

If we had access to cardinal utilities and a subjective probability were available, we could have used expected value as a basis for comparison of choices. However, these tools are not always available. Suppose for instance, that a voter has clear preferences among three candidates A, B and C. She prefers A to B to C. She still might not have a clear intuition in response to the question, "Do you prefer B or a 50-50 chance of A versus C?" [2]

The general issue is that a player in uncertainty is choosing between two sets (or sequences) of payoffs. The payoffs with L are, say, $a_1, a_2, ..., a_k$ and the payoffs with R are $b_1, b_2, ..., b_m$ such that $a_1 > a_2 > ... > a_k$ and $b_1 > b_2 > ... > b_m$. A conservative player chooses L over R if a_k is preferred to b_m. An aggressive player chooses R over L if b_1 is better than a_1. In addition to conservative and aggressive players, we can also consider *moderate* players who try to find the middle way, staying away from the maximum or the minimum.

More generally, let a player use a selection function f to represent a sequence of outcomes by a single element. f takes a finite ordered set as input and selects a representative element from that set. A conservative player uses the minimum, an aggressive player uses the maximum, and a moderate player uses (say) the median. (In case the number of elements is even we can use the higher of the two medians.)

If the selection function f is used then we let $X \preceq Y$ ($Y \succeq X$) iff $f(X) \leq f(Y)$, where X, Y are finite sets of outcomes. It is easily seen that the relation $\preceq$ defined this way will be transitive, although it is not anti-symmetric.

The function f should satisfy some rationality conditions.

1.1 Suitable Selection Functions

Definition 1. *A function f is* suitable *if it has the properties below.*

[2] Such choices between bets have been used by both Ramsey and de Finetti [9,4], but they do not always make sense to the person being asked. And without clear and consistent answers to such questions, we cannot have access to subjective probabilities (or to cardinal utilities). See also Savage [11] who imposes rationality conditions on choices between bets in order to derive subjective probabilities and utilities.

1. *If ordered sets A and B are isomorphic by an order preserving map g, then $g(f(A)) = f(B)$.*
2. *If set A is augmented to a set B by adding an element x which exceeds all elements of A, then $f(B) \geq f(A)$.*
3. *If set A is augmented to a set B by adding an element x which is less than all elements of A, then $f(B) \leq f(A)$.*
4. *If sets A and B overlap but all elements in $B - A$ exceed those in $A \cap B$ which exceeds all elements in $A - B$ then $f(A) \leq f(B)$.*

Note that conditions 2 and 3 together imply (for finite sets) condition 4 which implies both 2 and 3.

Let us define $s(n) = f(\{1, ..., n\})$.

Because of the order isomorphism property 1 above, the function s completely characterizes f. For any finite ordered set is order isomorphic to some set {1,...,n}. By abuse of language we will say that s is suitable iff the corresponding f is.

Proposition 1. *s is suitable iff $s(n) \leq s(n+1) \leq s(n)+1$ for all n.*

Proof

Necessity: Note that $f(\{1, ..., n\}) \leq f(\{1, ..., n+1\})$ since the second set is obtained from the first by adding a larger element. This yields $s(n) \leq s(n+1)$.

Note again that $f(\{1, ..., n+1\}) \leq f(\{2, ..., n+1\})$ since the first set is obtained from the second by adding a smaller element. But the second value is just $s(n)+1$. This follows from the isomorphism condition as the set $\{2, ..., n+1\}$ is isomorphic to the set $\{1, ..., n\}$ via the function $g(n) = n+1$. So we get $s(n+1) \leq s(n)+1$.

Before proving sufficiency we remark that this yields $s(n+m) \leq s(n)+m$. This is easily shown using induction on m.

Sufficiency: Suppose that s satisfies $s(n) \leq s(n+m) \leq s(n)+m$ for all m, n. It is easy to see that the first three conditions above will hold for the corresponding f.

To see the last, suppose that set A is $\{1, ..., n\}$ and set B is $\{m+1, ..., n, ..., m+r\}$ so that the elements $1, ..., m$ are in A and below B, and elements $n+1, ..., m+r$ are elements of B above A. Since we assume overlap as in property 4, we may assume that $m+r$ is at least n.

Now $f(B) = s(r)+m$ since B consists of r elements *in order* but shifted rightward from $1, ..., r$ by an amount of m. Since $n \leq r+m$, $s(n) \leq s(r+m) \leq s(r)+m$. Now $f(A) = s(n)$, and $f(B) = s(r)+m$. So indeed $f(A) \leq f(B)$. □

This shows that there are uncountably many suitable choice functions since the transition from $s(n)$ to $s(n+1)$ can be zero or one, infinitely many times. Since there are not that many human beings, the conservative humans with $s(n) = 1$, aggressive humans with $s(n) = n$ and moderate humans with $s(n)$ approximately equal to $n/2$ are the typical cases to appear in practice.

Lemma 1. *The minimum, the median and the maximum are all suitable functions (SCFs) in the sense above (and the corresponding notions of f-rationality are equivalent to being conservative, moderate, and aggressive respectively).*

Proof of Lemma 1: It is obvious that the median, the maximum and the minimum are preserved by isomorphism. We check the fourth condition in Definition 1 just for the median. Suppose that X and Y overlap so that X is $a_1 > a_2 > ... > a_k > b_1 > ... > b_m$ and Y is $b_1 > b_2 > ... > b_m > c_1 > ... > c_p$. $X - Y$ is above $Y - X$. Clearly if the median of X is an a_i or the median of Y is a c_i then we are done. If both medians are b_i and b_j respectively. Then $i + k = m - i + 1$ and $j = p + m - j + 1$. Thus we get $2i = m + 1 - k$ and $2j = p + m + 1$. Thus $i < j$ and $b_i > b_j$. □

Note that we could also conclude this from the last proposition since the median $m(n)$ on $\{1, ..., n\}$ obviously satisfies the condition $m(n) \leq m(n+1) \leq m(n)+1$ for all n. Ditto for the quartiles etc.

Note that an SCF need not satisfy Nash's IIA condition [7] that if $a = f(X)$, $Y \subseteq X$, and $a \in Y$ then $a = f(Y)$. It so happens that both the maximum and the minimum do satisfy this condition, but not the median.

Of course there is no particular reason why IIA *should* be obeyed in such a case. The role of $f(X)$ is to play the role of an element which in some sense *represents* X rather than that of a *most preferred* element of X. Thus the median is probably the closest to the expected value which we tend to use when we do have cardinal utilities and a subjective probability.

Definition 2. *Given an SCF f, an f-rational agent is an agent who, when uncertain between sets X and Y of alternatives, always picks X if $f(X) > f(Y)$.*

Sometimes we can speak of one strategy being dominated by another. Suppose that there is a set of possible worlds $\{s_1, ..., s_n\}$ and for each world s_i actions L and R yield payoffs p_i and q_i respectively. Then we say that L is dominated by R if for each i, $p_i \leq q_i$ and for at least one i, $p_i < q_i$.[3] In such a situation, the number of possible outcomes with either L or R will be the same, namely n.

It is easily seen that all three kinds of players, conservative, moderate and aggressive will never pick a strictly dominated strategy.

[3] The notion of *strictly dominated* strategy that we intend is *subjective*. Thus we actually mean a strategy which is dominated by another pure strategy that the agent *considers possible* at the time of play. Consider the following scenario: Suppose Ann is giving a dinner party and she has to make a decision between serving two dinners d and d'. Assume that three guests have been invited, and guest 1 and guest 2 prefer d to d' whereas guest 3 not only prefers d' to d but also, he is allergic to one of the ingredients in d. If Ann is a conservative agent, she will serve d' to avoid the worst case scenario. Now let's say guest 3 is not going to the dinner party but Ann is unaware of this. In this case she will still serve d' even though it is a strictly dominated strategy if only guests 1 and 2 will be there. But since guest 3 was expected to show up in the original scenario and Ann does not know that he is not coming, we do not consider serving d' as strictly dominated in this example. Note that the notion of a dominated strategy makes sense only when two choices lead to different results but over the same set of possible worlds. When one is choosing between L and R in a game tree, the two actions lead to different sets of nodes and the notion of domination does not have an obvious intuitive meaning.

2 Examples

We now give some applications of the technical work so far.

2.1 Example 1

The Asian Disease – Tversky and Kahneman 1981 [13]
Imagine that the United States is preparing for the outbreak of an unusual Asian disease, which is expected to kill 600 people. Two alternative programs to combat the disease have been proposed. Assume that the exact scientific estimates of the consequences of the programs are as follows:

- If Program A is adopted, 200 people will be saved.
- If Program B is adopted, there is a one-third probability that 600 people will be saved and a two-thirds probability that no people will be saved.

Which of the two programs would you favor?
In this version of the problem, a substantial majority (72%) of 152 respondents favor program A, indicating risk aversion.

Other 155 respondents, selected at random, receive a question in which the same cover story is followed by a different description of the options:

- If Program A′ is adopted, 400 people will die.
- If Program B′ is adopted, there is a one-third probability that nobody will die and a two-thirds probability that 600 people will die.

Again, the same question is asked:
Which of the two programs would you favor?
A clear majority (78%) now favor B′.

One way to understand this phenomenon is that the way the problem is stated causes a change from *min*-rationality to *max*-rationality. Programs A and A′ result in 400 deaths. Programs B and B′ amount to no deaths if we are lucky and to 600 deaths if we are unlucky. A cautious person would prefer program A and an aggressive (optimistic) person would prefer B. The way the question is posed causes a shift from caution to optimism. But both forms of rationality are 'rational'.

Rationality can enter at two different levels. A purely decision theoretic level, where a single agent is trying to make the 'best' choice; or a game theoretic level where two or more agents are involved, and we not only have to think about the 'rational' choice but also about what another agent thinks is the rational choice.

The previous example was in a purely decision theoretic mode. We now consider how the notion of 'temperament-based' rationality can be helpful in understanding two classic game theoretic examples. We consider as our first example an agent (the husband) in a state of uncertainty and show how the play is affected by his temperament.

2.2 Example 2

In the Bach-Stravinsky game [8], the couple want to go to a concert but the wife (row) prefers Bach and the husband prefers Stravinsky. But each would rather go together than listen to their favourite composer by themselves.

Here are the payoffs. Row's payoffs in each box are listed first.[4]

	Bach	Stravinsky
Bach	3, 2	0, 0
Stravinsky	1, 1	2, 3

Fig. 1.

If they go to different events they are not happy so the payoffs are low for both. If they go to the *same event*, then both have positive payoffs, but the wife's is higher if they go to the Bach and the husband's is higher if they go to Stravinsky. There are two Nash equilibria, the Bach-Bach one with payoffs (3,2), and the Stravinsky-Stravinsky one with (2,3).

Figure 2 presents this as an extensive form game with the wife choosing first and the husband next, but we leave it open whether the husband knows the wife's choice.

We consider various scenarios involving the husband's knowledge and temperament. We assume that the wife knows the husband's payoffs and temperament and he does not know hers.

Case 1) Husband does not know wife's move (and she knows this).

a) He is aggressive. Then being aggressive, he will choose S (Stravinsky) for his move since the highest possible payoff (for him) is 3. Anticipating his move, she will also choose S, and they will end up with payoffs of (2,3).

b) The husband is conservative. Then not knowing what his wife chose, he will choose B since the minimum payoff of 1 is better than the minimum payoff of 0. Anticipating this, the wife will also choose B and they will end up with (3,2).

Case 2) Finally if the husband *will* know what node he is at (and the wife knows this), then the wife, regardless of the husband's temperament, will choose B. The husband will also choose B and they will end up at (3,2).

2.3 Example 3

The centipede game (Figure 3) was introduced by Rosenthal in [10]. The game starts at node A, with player 1 choosing his move: *across* or *down*. If he chooses *down*, the game ends. If he chooses *across*, it will be player 2's turn to choose

[4] We have made the payoffs for Bach-Stravinsky and Stravinsky-Bach different so that the extensive form is generic.

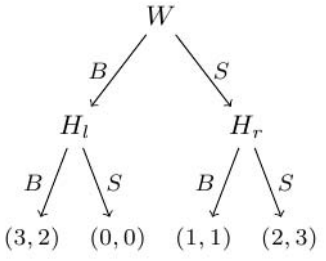


Fig. 2.

at node B. If player 2 chooses *across* at B, the game will continue to node C where player 1 will choose a move again and so on. The numbers in brackets next to the nodes indicate the player that will play at that node. The first payoff belongs to player 1 and the second to player 2.

Aumann showed, using backward induction and common knowledge of rationality that player 1 would choose *down* at node A, ending the game and forgoing the higher payoffs later on. [2]

The argument is roughly as follows. *If* the game ever reached node E, then 1 would choose *down*, getting 6 rather than *across*, getting only a 5. This choice by 1 would yield 2 a payoff of 5. But then at D, player 2, anticipating 1's move at E, and knowing that 1 is rational would say, "Why should I choose *across* and get 5 when I can choose *down* right now and get 6?" Thus at D, 2 would choose *down* and 1 would only get 3. Working backward this way from E to D to C to B to A, we get the conclusion that at node A, 1 would choose *down*.

But this argument requires backward induction, and deep consideration by both 1 and 2 of each other's mindsets. Can this be avoided?

Indeed Artemov [1] uses a simpler method to derive the same result as Aumann. Artemov defines a rational player as one who makes a decision based on the "highest *guaranteed* payoff", subject to the player's knowledge. In other words he describes as rational the kind of player we have chosen to call conservative.

He then shows (his Theorem 1) that in the centipede game, a rational player *in his sense* will follow the backward induction solution *even in the absence of common knowledge of rationality*. This follows easily from the observation that at each node, the unique payoff (for the player whose turn it is) from choosing *down* is higher than the lowest possible payoff from going *across*.

Thus Artemov generalizes Aumann's result, replacing common knowledge of rationality by plain rationality.[5]

Now consider a *moderate* player playing this game. If he had been conservative and used backward induction, then as we saw he would choose *down* at once and

[5] Artemov's argument applies only to the centipede game. For other games Artemov's solution could diverge from the backward induction solution.

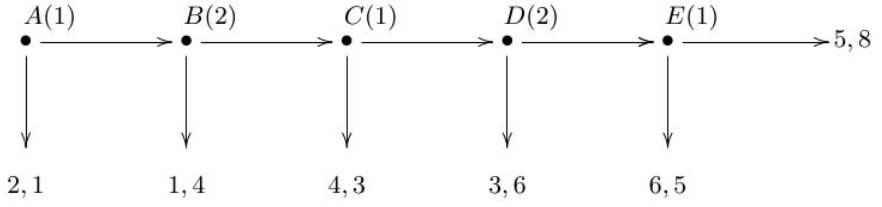


Fig. 3.

get a certain payoff. But since he is a moderate, he will see that the median from playing *across* is much higher. If both players are moderate players, and agnostic about the rest of the game, then the game will continue for quite a while, with both players choosing *across* and earning much larger payoffs. Thus our notion of a moderate player shows the rationality of the common pattern seen in ordinary behaviour (see [6] for instance) where players play *across* for quite a while.

3 Comparison with Previous Work

The issue we have discussed so far is: if a player has preferences among *elements* of a certain ordered set, how does that preference translate to preferences among *subsets?* In the previous discussion we used a selection function f to convert an ordering between elements to an ordering between sets. But the problem has been looked at more abstractly without relying on a selection function.

Suppose for instance that a voter has preferences among candidates for an election, how will that translate to preferences among slates of candidates? Two conditions which relate preferences among individual elements to preferences among subsets are discussed in [5,3].

Let R be a linear order on a set X and let P be the strict order part of R, i.e. xPy iff $xRy \wedge \neg yRx$. Let Ξ be the set of all finite subsets of X and let $\succeq$ be an order on Ξ. $\succ$ is the strict part of $\succeq$. For singleton sets, we assume that $\{x\} \succeq \{y\}$ iff xRy. So singletons follow their (unique) elements. [5,3] discuss two conditions on R and $\succ$.

Dominance: For all $A \in \Xi$ and $x \in X$

(i) $[xPy$ for all $y \in A] \rightarrow A \cup \{x\} \succ A$

(ii) $[yPx$ for all $y \in A] \rightarrow A \succ A \cup \{x\}$

Independence: For all $A, B \in \Xi$, for all $x \in X - A \cup B$

$$A \succ B \rightarrow A \cup \{x\} \succeq B \cup \{x\}$$

(Note: we are using the symbol "–" also for set subtraction.)

The conditions Dominance and Independence are called (G) and (M) by Kannai and Peleg [5] who showed that if X has at least six elements then these very natural conditions are incompatible.

If we use the technique of comparing the minimum, the maximum or the median as ways of comparing sets, then all three techniques satisfy a weaker form of the dominance condition.

Weak Dominance: For all $A \in \Xi$ and $x \in X$

$$\text{(i) } [xPy \text{ for all } y \in A] \rightarrow A \cup \{x\} \succeq A$$
$$\text{(ii) } [yPx \text{ for all } y \in A] \rightarrow A \succeq A \cup \{x\}$$

For instance, suppose we say that $A \succeq B$ iff $min(A) \geq min(B)$ then this definition of $\succeq$ satisfies both independence and weak dominance. max also obeys independence and weak dominance.

However, independence fails all SCFs except the min and the max. Suppose that there is an n such that $1 < s(n) < n$. It follows that there is an n such that $s(n) < s(n+1) = s(n)+1 < n+1$. (Take the smallest n such that $s(n) < s(n+1)$). Let $m = s(n)$ and $m + 1 = s(n + 1)$.

To simplify notation, say $n = 5$, $m = 2$. Let $X = \{1, 5, 6, 8, 9\}$ and $Y = \{1, 4, 7, 8, 9\}$. Then clearly f will pick 5 from X and 4 from Y so X will be greater than Y.

Add 10 to both sets. $X' = \{1, 5, 6, 8, 9, 10\}$ and $Y' = \{1, 4, 7, 8, 9, 10\}$. Now f will pick 6 from X' and 7 from Y' so Y' will be greater than X'.

3.1 Seidenfeld's Result

Can we justify choices based on ordinal utilities as choices made on the basis of unknown cardinal utilities and subjective probabilities? In other words, can we take the point of view that if we knew the cardinal utitlies and subjective probabilities then we would make the same choices which our principles justify?

Suppose[6] that the set of possible outcomes of an act are identified by their ordinal ranks, i.e. f identifies the ordinally ranked possible outcomes of an act A_i: $f(A_i) = \{r_{i,1}, ..., r_{i,n}\}$ where for convenience, $r_{i,j} < r_{i,k}$ iff $j < k$. We assume that these are *increasing* ordinal ranks.

Say that one set of ordinally ranked outcomes $\{r_1, ..., r_j\}$ is *strictly inferior* to another set $\{r_k, ..., r_n\}$ if $r_j < r_k$, i.e., if all the possible outcomes in the first set are ordinally dispreferred to all of those in the second set. Write this as

$$\{r_1, ...r_j\} \sqsubset \{r_k..., r_n\}$$

Given two acts/options $\{A_1, A_2\}$ write $C_{A_1,A_2} = f(A_1) \cap f(A_2)$, those possible outcomes they have in common.

Now to restate the "suspect" ordinal decision rule (independence).

$$\text{If } f(A_1) - C_{A_1,A_2} \sqsubset f(A_2) - C_{A_1,A_2} \text{ then } A_1 \prec A_2$$

Proposition 2. *There is* no *Bayes model for this decision rule. That is, there is no consistent assignment of probabilities to states of uncertainty such that for* all

[6] The following result about condition 4 of definition 1 was proved by Teddy Seidenfeld [12] on the first author's birthday after hearing his talk at CMU on November 20, 2011. It is included here with his permission. His A_1 is our A and his A_2 is our B when comparing with condition 4 of definition 1.

cardinal utilities consistent with the ordinal ranks, $\prec$ agrees with a comparison of SEU (subjective expected utility).

The proposition follows from the following lemma.

Lemma 2. *Consider a family of acts defined (measurable) with respect to a common k-fold partition $\Omega_k = \{\omega_1, ..., \omega_k\}$. Then the only Bayes models consistent with the suspect rule applied to this family require $p(\omega_j) = 1/k$ $(j = 1, ..., k)$ a uniform distribution over Ω_k.*

The proposition follows by considering two families of acts. The first family is defined with respect to Ω_k. The second family is defined with respect to a coarsened partition Ω'_{k-1} where $\omega'_i = \omega_i : i \leq k-2$ and $\omega'_{k-1} = \{\omega_{k-1}, \omega_k\}$. That is, Ω'_{k-1} is obtained by coarsening the last two elements of Ω_k.

By the lemma, from the first family, $p(\omega_1) = 1/k$, and from the second family $p(\omega'_1) = 1/(k-1)$. But $\omega'_1 = \omega_1$.

To get an intuition about the lemma[7], suppose that $k = 2$ and the states ω_1, ω_2 have probabilities .6 and .4 respectively. Let act A have outcomes 10 and 5 in states ω_1, ω_2 respectively, and let act B have outcomes 5 and 11 in the two states. Then the suspect principle says that act B should be preferred. However, the expected utility of act A is $6 + 2 = 8$ and that of B is $3 + 4.4 = 7.4$ which is lower. Thus condition 4, definition 1, *can* force us to go against expected utility if the probabilities of the various outcomes are not equal.

The lemma is established by considering acts such as:

	ω_1	ω_2	...	ω_{k-2}	ω_{k-1}	ω_k
A_1^k	1	2	...	$k-2$	$k-1$	k
A_2^k	1	2	...	$k-2$	$k+1$	$k-1$

So $C_{A_1^k, A_2^k} = \{1, ..., k-1\}$,
$f(A_1^k) - C_{A_1^k, A_2^k} = \{k\}$, and
$f(A_2^k) - C_{A_1^k, A_2^k} = \{k+1\}$.

Hence, $A_1 \prec A_2$ by the suspect rule, since $\{k\} \sqsubset \{k+1\}$.

The only Bayes models for the "ordinal" preference that are invariant over all conditional utilities consistent with these ordinal ranks require $p(\omega_{k-1}) \geq p(\omega_k)$.

By considering automorphisms on Ω_k and the induced automorphisms on A_1^k and A_2^k , we get

$$p(\omega_i) \geq p(\omega_j) \ \forall i, j \text{ such that } 1 \leq i, j \leq k$$

That is, then $p(\omega_i) = 1/k$, $i = 1, ..., k$ as claimed.

Seidenfeld's result indicates a certain tension between choices based on purely ordinal utilities and the idea that ordinal choices should be consistent with possible (but unknown) cardinal utilities.

[7] This particular paragraph is not due to Seidenfeld, but is furnished by us, PTW, to help along the reader's intuition.

4 Concluding Remarks

We have reviewed some classical results on the transfer of choices from points to sets of points and suggested some ways in which an agent might act, who lacks an access to cardinal utilities and probabilities, but does know how she would choose between pairs of points. Various classical results, e.g. by Kannai and Peleg, and the new one by Seidenfeld show that the task is not free of problems. But it is a real task nonetheless. The world did not stop having elections when Arrow's theorems were proved. In conclusion, one must do the best one can with a defective tool.

We end with a general remark about rationality which is especially relevant when logical omniscience is not pre-supposed.

Three Aspects of Rationality

1. Making the best of the choices one believes oneself to have.
2. Fully using logic to eliminate impossible choices.
3. Using psychology to predict the choices of other people.

Perhaps we should keep these three aspects separate and reason about them separately.

Acknowledgements. We thank Pradeep Dubey, Sambuddha Ghosh, R. Ramanujam, Yair Tauman, Shumel Zamir and two anonymous referees for helpful comments. The research of the first author was partially supported by a grant from CUNY-FRAP.

References

1. Artemov, S.: Rational decisions in non-probabilistic settings, technical report TR-2009012, CUNY Ph.D. program in Computer Science (2009)
2. Aumann, R.: Backward induction and common knowledge of rationality. Games and Economic Behavior 8, 6–19 (1995)
3. Barbera, S., Bossert, W., Pattanaik, P.: Ranking sets of objects, University of Montreal, Cahier (2001-2002), `papyrus.bib.umontreal.ca`
4. de Finetti, B.: On the subjective meaning of probability (Sul significato soggettivo della probabilita). Fundamenta Mathematicae 17, 298–329 (1931)
5. Kannai, Y., Peleg, B.: A note on the extension of an order on a set to the power set. J. Economic Theory 32, 172–175 (1984)
6. McKelvey, R., Palfrey, T.: An Experimental Study of the Centipede Game. Econometrica 60, 803–836 (1992)
7. Nash Jr., J.F.: The Bargaining Problem. Econometrica 18, 155–162 (1950)
8. Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press (1994)
9. Ramsey, F.P.: Foundations: Essays in Philosophy, Logic, Mathematics and Economics. Humanities Press (1977) (originally published in 1931)
10. Rosenthal, R.: Games of Perfect Information, Predatory Pricing, and the Chain Store. Journal of Economic Theory 25, 92–100 (1981)

11. Savage, L.: The Foundations of Statistics. Wiley (1954)
12. Seidenfeld, T.: Personal Communication
13. Tversky, A., Kahneman, D.: The Framing of Decisions and the Psychology of Choice. Science 211, 453–458 (1981)

The Complexity of Inhabitation with Explicit Intersection

Jakob Rehof[1] and Paweł Urzyczyn[2,⋆]

[1] Technical University of Dortmund
Department of Computer Science
jakob.rehof@cs.tu-dortmund.de
[2] University of Warsaw
Institute of Informatics
urzy@mimuw.edu.pl

Abstract. It is shown that the inhabitation problem for intersection types without the intersection introduction rule is Expspace-complete and that the further restriction without subtyping is Pspace-complete.

1 Introduction

We consider the complexity of the inhabitation problem (is there a pure λ-term that can be assigned a given type?) for the λ-calculus with intersection types [4] under the restriction that the intersection introduction rule ($\cap$I) is eliminated. We refer to the restricted system as $\lambda(-\cap\mathrm{I})$. Whereas the inhabitation problem for the full intersection type system is undecidable [16], it was shown by Kurata and Takahashi [7] that the inhabitation problem for $\lambda(-\cap\mathrm{I})$ is decidable. However, the question of complexity of the inhabitation problem for $\lambda(-\cap\mathrm{I})$ was left open. We prove here that the problem is Expspace-complete. In addition, we show that the further restriction without subtyping is Pspace-complete.

The system $\lambda(-\cap\mathrm{I})$ is a natural restriction of the full intersection type system [4], because it is equivalent to the *explicitly* typed version ("Church style fragment") of that system [7, Lemma 3.1]), in which all abstractions are of the form $\lambda x:\tau.M$. One can also say that in system $\lambda(-\cap\mathrm{I})$ intersections are explicit because, in the absence of intersection introduction, they cannot be introduced by the typing rules.

The full intersection type system arises from the Curry-style simply-typed λ-calculus [2] by addition of the rules

$$\frac{\Gamma\vdash M:\tau_1 \quad \Gamma\vdash M:\tau_2}{\Gamma\vdash M:\tau_1\cap\tau_2}(\cap\mathrm{I}) \quad \frac{\Gamma\vdash M:\tau_1\cap\tau_2}{\Gamma\vdash M:\tau_i}(\cap\mathrm{E}) \quad \frac{\Gamma\vdash M:\tau \quad \tau\le\tau'}{\Gamma\vdash M:\tau'}(\le)$$

where the subtyping rule ($\le$) refers to a certain quasi-order on types. Usually, the intersection elimination rule ($\cap$E) is contained in the subtyping theory, by the axiom $\tau_1\cap\tau_2\le\tau_i$, and consequently rule ($\cap$E) is left out in systems with

⋆ Partly supported by MNiSW grant N N206 355836.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 256–270, 2012.

subtyping thus presented (we follow this convention in the present paper). However, in the standard core system without subtyping, the rule ($\cap$E) is present. In the present paper, we consider the system $\lambda(-\cap\mathrm{I})$, where ($\cap$I) is eliminated (and ($\cap$E) is contained in rule ($\leq$)) as well as the system $\lambda(-\cap\mathrm{I})$ without subtyping, in which ($\cap$E) is present and both of the rules ($\cap$I) and ($\leq$) are absent.

Intersection type systems belong to a class of propositional logics with enormous expressive power. Intersection types capture several semantic properties of λ-terms, and their type reconstruction problem has long been known to be undecidable, since they characterize exactly the set of strongly normalizing terms [11]. The inhabitation problem for the full system is undecidable [16] and is closely related to the λ-definability problem [13].

Our result adds to a recently growing body of knowledge concerning the fine structure of the intersection type inhabitation (provability) problem obtained by considering various restrictions of the system. Recall that the borderline between decidability and undecidability was clarified in [8,17] by rank restrictions (as defined in [9]), with inhabitation in rank 2 types being shown EXPSPACE-complete and undecidable from rank 3 and up. The system without intersection elimination ($\cap$E) was shown to be decidable in [18] (the upper bound is non-elementary and the exact complexity is open). In finite combinatory logic with intersection types [12] (combinatory logic with intersection types [5] restricted to monomorphic types) inhabitation is EXPTIME-complete with or without subtyping. Finite combinatory logic with intersection types can be presented as the restriction of system $\lambda(-\cap\mathrm{I})$, in which the function type introduction rule ($\to$ I) is eliminated. Seen in this perspective, the present result allows us to compare the relative complexity of the explicitly typed rule ($\to$ I) in terms of inhabitation complexity (EXPTIME vs. EXPSPACE). Following [12], inhabitation can be seen as a foundation for type-based function composition synthesis. Under this perspective, our result determines the complexity of synthesis in the "Church-style" fragment of the intersection typed λ-calculus. Finally, our present result shows that, in contrast to some other restrictions (e.g., finite combinatory logic), the presence of subtyping makes a big difference (EXPSPACE vs. PSPACE). Intuitively, in system $\lambda(-\cap\mathrm{I})$, subtyping can express a certain part of the logic contained in intersection introduction. For example, the judgment

$$f : (p \to q) \cap (r \to s), \quad x : p \cap r \quad \vdash \quad fx : q \cap s$$

is derivable in the full system without subtyping (because $fx : q$ and $fx : s$ can be derived separately), and also in system $\lambda(-\cap\mathrm{I})$ with subtyping (because $(p \to q) \cap (r \to s) \leq p \cap r \to q \cap s$), but not in system $\lambda(-\cap\mathrm{I})$ without subtyping.[1]

The situation is summarized in the following overview, where we list the known results for the various restrictions mentioned above. In addition to the rank restrictions we consider the cases where one or more of the rules ($\cap$I), ($\cap$E), ($\leq$), ($\to$I) are eliminated (indicated by a "$-$"). In the cases where rule ($\leq$) is not eliminated, the presence or absence of rule ($\cap$E) is immaterial.

[1] However, subtyping and ($\cap$I) are in general incomparable: for instance the judgment $\vdash \mathbf{I} : (p \to p) \cap (q \to q)$ requires ($\cap$I), while $x : p \to q \cap r \vdash x : p \to q$ requires ($\leq$).

Rank $k = 2$	:	EXPSPACE-complete
Rank $k > 2$	:	Undecidable
$-(\cap\mathrm{I})$	:	EXPSPACE-complete
$-(\cap\mathrm{I}, \leq)$	:	PSPACE-complete
$-(\cap\mathrm{E}, \leq)$	:	Decidable
$-(\to\mathrm{I})$	:	EXPTIME-complete
$-(\to\mathrm{I}, \leq)$	:	EXPTIME-complete

Let us note here that *alternation* [3] plays a fundamental role in problems of type inhabitation. Intuitively, we may need to consider subgoals of the form $\Gamma \vdash ? : \tau$ where the inhabitant (indicated by "?") is an application of the form $(xY_1 \ldots Y_n)$. This is equivalent to asking whether there *exists* a variable x in Γ with a type of the form $\tau_1 \to \cdots \to \tau_n \to \tau$ such that *all* of the types τ_1 through τ_n are inhabited. Hence, one can cast the PSPACE-algorithm for inhabitation in simple types [14] as an alternating polynomial time procedure (see also [15]). In intersection type systems, inhabitation goals may involve questions of the form $\Gamma \vdash ? : \tau_1 \cap \tau_2$ leading to the parallel (universal) questions of inhabitation in τ_1 *and* τ_2 by *some* single term X. In the present paper, our exponential space upper bound is achieved by an alternating exponential time procedure, and our lower bound relies on the computational model of *bus machines*, introduced in [17], which generically simulates alternating exponential time Turing machines.

Comparing the EXPSPACE-completeness of inhabitation with rank 2 types [17] to our present result, it is remarkable that the same model (bus machines) is used as a basis of reduction in the hardness proofs in both cases. However, the reduction must be adjusted, as a result of fundamental differences in the systems under consideration. In the former case, types are restricted (rank 2) and the typing rules are unrestricted, whereas here we consider unrestricted types with restricted rules (see Section 3).

2 Preliminaries

Types: Type expressions, ranged over by τ, σ etc., are defined by

$$\tau \ ::= \ a \mid \tau \to \tau \mid \tau \cap \tau$$

where $a, b, c, \ldots$ range over *atoms* comprising of type constants, including the constant ω, and type variables. As usual, types are taken modulo commutativity $(\tau \cap \sigma = \sigma \cap \tau)$, associativity $((\tau \cap \sigma) \cap \rho = \tau \cap (\sigma \cap \rho))$, and idempotency $(\tau \cap \tau = \tau)$. A type *environment* Γ is a finite set of type assumptions of the form $x : \tau$, and we let $dm(\Gamma)$ and $rn(\Gamma)$ denote the domain and range of Γ.

A type $\tau \cap \sigma$ is said to have τ and σ as *components*. For an intersection of several components we sometimes write $\bigcap_{i=1}^{n} \tau_i$ or $\bigcap_{i \in I} \tau_i$ or $\bigcap\{\tau_i \mid i \in I\}$, where the empty intersection is identified with ω.

If $\tau = \tau_1 \to \cdots \to \tau_n \to \sigma$, then we write $\sigma = tgt_n(\tau)$ and $\tau_i = arg_i(\tau)$, for $i \leq n$. A type of the form $\tau_1 \to \cdots \to \tau_n \to p$, where $p \neq \omega$ is an atom, is called

a *path of length n*. A type τ is *organized* if it is a (possibly empty) intersection of paths (those are called *paths in* τ). Note that premises in an organized type do not have to be organized, i.e., organized is not necessarily normalized [6].

Subtyping: Subtyping $\leq$ is the least quasi-order (reflexive and transitive relation), satisfying the following conditions:

$$\sigma \leq \omega, \quad \omega \leq \omega \to \omega, \quad \sigma \cap \tau \leq \sigma, \quad \sigma \cap \tau \leq \tau, \quad \sigma \leq \sigma \cap \sigma;$$

$$(\sigma \to \tau) \cap (\sigma \to \rho) \leq \sigma \to \tau \cap \rho;$$

$$\text{If } \sigma \leq \sigma' \text{ and } \tau \leq \tau' \text{ then } \sigma \cap \tau \leq \sigma' \cap \tau' \text{ and } \sigma' \to \tau \leq \sigma \to \tau'.$$

We identify σ and τ when $\sigma \leq \tau$ and $\tau \leq \sigma$. Note that $\tau \to \omega = \omega$, for all τ. The distributivity properties below follow from the axioms of subtyping:

$$(\sigma \to \tau) \cap (\sigma \to \rho) = \sigma \to (\tau \cap \rho)$$

$$(\sigma \to \tau) \cap (\sigma' \to \tau') \leq (\sigma \cap \sigma') \to (\tau \cap \tau')$$

The following property, probably first stated in [1], is often called *beta-soundness*. Note that the converse is trivially true.

Lemma 1. *Let a_j, for $j \in J$, be atoms.*

1. *If $\bigcap_{i\in I}(\sigma_i \to \tau_i) \cap \bigcap_{j\in J} a_j \leq \alpha$ then $\alpha = a_j$, for some $j \in J$.*
2. *If $\bigcap_{i\in I}(\sigma_i \to \tau_i) \cap \bigcap_{j\in J} a_j \leq \sigma \to \tau$, where $\sigma \to \tau \neq \omega$, then the set $\{i \in I \mid \sigma \leq \sigma_i\}$ is nonempty and $\bigcap\{\tau_i \mid \sigma \leq \sigma_i\} \leq \tau$.*

Lemma 2. *Every type τ has an equivalent organized type $\overline{\tau}$, computable in polynomial time.*

Proof: Define $\overline{a} = a$ if a is an atom, and $\overline{\tau \cap \sigma} = \overline{\tau} \cap \overline{\sigma}$. If $\overline{\sigma} = \bigcap_{i\in I} \sigma_i$ then take $\overline{\tau \to \sigma} = \bigcap_{i\in I}(\tau \to \sigma_i)$. □

In what follows types are assumed to be organized.

Lemma 3. *Let $\bigcap_{i\in I} \tau_i \leq \beta_1 \to \cdots \to \beta_n \to p$, where τ_i are paths. Then there is an $i \in I$ such that $\tau_i = \alpha_1 \to \cdots \to \alpha_n \to p$ and $\beta_j \leq \alpha_j$, for all $j \leq n$.*

Proof: Induction with respect to n, using the beta soundness (Lemma 1). □

Following [7], we define the least upper bound $\sigma_1 \oplus \cdots \oplus \sigma_n$ of organized types $\sigma_1, \ldots, \sigma_n$ as the intersection of all paths of the form

$$(\alpha_1^1 \cap \cdots \cap \alpha_n^1) \to \cdots \to (\alpha_1^k \cap \cdots \cap \alpha_n^k) \to p,$$

where $p \neq \omega$ is an atom and, for all $i \leq n$, type $\alpha_i^1 \to \cdots \to \alpha_i^k \to p$ is a path in σ_i. It should be obvious that $\sigma_i \leq \sigma_1 \oplus \cdots \oplus \sigma_n$, for all $i \leq n$. We show that it is actually the l.u.b.:

Lemma 4. *If $\sigma_i \leq \tau$, for all $i \leq n$, then $\sigma_1 \oplus \cdots \oplus \sigma_n \leq \tau$.*

Proof: We prove the claim for $\tau = \tau^1 \to \cdots \to \tau^k \to p$ being a path. (The general case then follows easily.) By Lemma 3 there are paths $\alpha_i^1 \to \cdots \to \alpha_i^k \to p$ in σ_i such that $\tau^j \leq \alpha_i^j$, for all i, j. Then in $\sigma_1 \oplus \cdots \oplus \sigma_n$ we have a path $\pi = (\alpha_1^1 \cap \cdots \cap \alpha_n^1) \to \cdots \to (\alpha_1^k \cap \cdots \cap \alpha_n^k) \to p$. Since $\tau^j \leq \alpha_1^j \cap \cdots \cap \alpha_n^j$, we obtain $\pi \leq \tau$, whence $\sigma_1 \oplus \cdots \oplus \sigma_n \leq \tau$. □

Type Assignment

We consider the standard intersection type assignment system [4] for pure λ-terms without the rule for intersection introduction. The system is shown in Figure 1. As already noted, the system is equivalent to the explicitly typed version of the standard system (see [7]). We have the following generation lemma:

$$\frac{}{\Gamma, x:\tau \vdash x:\tau}(\text{var}) \qquad \frac{\Gamma \vdash M:\tau\to\tau' \quad \Gamma \vdash N:\tau}{\Gamma \vdash (M\ N):\tau'}(\to\text{E})$$

$$\frac{\Gamma, x:\tau \vdash M:\tau'}{\Gamma \vdash \lambda x\, M:\tau\to\tau'}(\to\text{I}) \qquad \frac{\Gamma \vdash M:\tau \quad \tau\le\tau'}{\Gamma \vdash M:\tau'}(\le)$$

Fig. 1. Type system $\lambda(-\cap\text{I})$

Lemma 5

1. *If $\Gamma \vdash MN:\tau$ then $\Gamma \vdash M:\sigma\to\tau$, and $\Gamma \vdash N:\sigma$, for some σ.*
2. *If $\Gamma \vdash xN_1\ldots N_k:\tau$ then $\Gamma(x)\le\sigma_1\to\cdots\to\sigma_k\to\tau$, where $\Gamma \vdash N_i:\sigma_i$, for all $i\le k$. In particular, $\Gamma \vdash x:\tau$ implies $\Gamma(x)\le\tau$.*
3. *If $\Gamma \vdash \lambda x\, M:\tau$ then there are σ,ρ with $\Gamma, x:\sigma \vdash M:\rho$ and $\sigma\to\rho\le\tau$.*

Proof: Routine. Type derivation ends with a logical rule followed by subsumption. In part 2 use induction. □

Subject reduction now follows easily:

Lemma 6. *If $\Gamma \vdash M:\sigma$ and $M \twoheadrightarrow_\beta M'$ then $\Gamma \vdash M:\sigma$.*

Proof: Standard induction using the substitution property: If $\Gamma, x:\tau \vdash M:\sigma$ and $\Gamma \vdash N:\tau$ then $\Gamma \vdash M[N/x]:\sigma$. □

System $\lambda(-\cap\text{I})$ has the strong normalization property, as a subsystem of the ordinary intersection type-assignment. It therefore follows from Lemma 6 that an inhabited type must have a normal inhabitant.

Lemma 7. *If $\Gamma \vdash M:\sigma$ then there is a normal form M' with $\Gamma \vdash M':\sigma$.*

The crucial Lemmas 9 and 11 below characterize types of normal forms. To prove them we need the following simple property:

Lemma 8. *If $\tau\le\sigma$ and $\Gamma, x:\sigma \vdash M:\rho$ then also $\Gamma, x:\tau \vdash M:\rho$.*

Proof: Routine induction. □

Lemma 9. *The following are equivalent conditions:*

1. $\Gamma \vdash \lambda x\, M : \bigcap_{i\in I}(\tau_i \to \sigma_i)$;
2. $\Gamma,\ x : \bigoplus_{i\in I} \tau_i \vdash M : \bigcap_{i\in I} \sigma_i$.

Proof: (1 ⇒ 2) By the generation lemma, we have $\Gamma, x:\tau \vdash M : \sigma$, with $\tau \to \sigma \leq \bigcap_{i\in I}(\tau_i \to \sigma_i)$. That is, $\tau_i \leq \tau$ and $\sigma \leq \sigma_i$, for every i. It follows that $\bigoplus_{i\in I} \tau_i \leq \tau$ (by Lemma 4) and $\sigma \leq \bigcap_{i\in I} \sigma_i$, and we use Lemma 8.

(2 ⇒ 1) If (2) then $\Gamma \vdash \lambda x\, M : \bigoplus_{i\in I} \tau_i \to \bigcap_{i\in I} \sigma_i$ and we apply subsumption using the inequality $\bigoplus_{i\in I} \tau_i \to \bigcap_{i\in I} \sigma_i \ \leq\ \bigcap_{i\in I}(\tau_i \to \sigma_i)$. □

Remark 10. Observe that Lemma 9 has the following "paradoxical" consequence: An abstraction of type $\bigcap_{i\in I}(\tau_i \to \sigma_i)$ must have type $\bigcap_{i,j\in I}(\tau_i \to \sigma_j)$. It is also not enough to know that $\Gamma, x:\tau_i \vdash M : \sigma_i$ holds for all $i \in I$ to conclude that $\Gamma \vdash \lambda x\, M : \bigcap_{i\in I}(\tau_i \to \sigma_i)$. The latter remains true even if all σ_i are the same: the conjunction of $x:\tau \vdash M : \sigma$ and $x:\rho \vdash M : \sigma$ does not imply $x : \tau\oplus\rho \vdash M : \sigma$. Indeed, we have $x : (p \to p) \to q \vdash x\mathbf{I} : q$ and $x : (r \to r) \to q \vdash x\mathbf{I} : q$. But, on the other hand, $((p \to p) \to q) \oplus ((r \to r) \to q) = (p \to p) \cap (r \to r) \to q$ and $x : (p \to p) \cap (r \to r) \to q \nvdash x\mathbf{I} : q$, because $\nvdash \mathbf{I} : (p \to p) \cap (r \to r)$.

Lemma 11 (Path Lemma). *The following are equivalent conditions:*

1. $\Gamma \vdash xN_1 \dots N_k : \tau$;
2. *There exists a set P of paths in $\Gamma(x)$ such that*
   (a) $\bigcap_{\pi\in P} tgt_k(\pi) \ \leq\ \tau$;
   (b) $\Gamma \vdash N_i : \bigcap_{\pi\in P} arg_i(\pi)$, *for all $i \leq k$.*

Proof: (1 ⇒ 2) Similar to the proof of Lemma 10 in [12]. By Lemma 5 we have $\Gamma(x) \leq \sigma_1 \to \dots \to \sigma_k \to \tau$, where $\Gamma \vdash N_i : \sigma_i$, for all $i \leq k$. Now, assume that $\Gamma(x)$ is organized and let ρ be the organized form of $\sigma_1 \to \dots \to \sigma_k \to \tau$. We apply lemma 3 to every path in ρ. This yields a set P of paths in $\Gamma(x)$ such that, for every path μ in ρ, there is $\pi \in P$ with $\sigma_i \leq arg_i(\pi)$ and also $tgt_k(\pi) \leq tgt_k(\mu)$. Since all paths in (organized) τ are of the form $tgt_k(\mu)$, we obtain $\bigcap_{\pi\in P} tgt_k(\pi) \leq \tau$ (part 2a). Part 2b follows by subsumption.

(2 ⇒ 1) Easy (as in [12]). □

3 EXPSPACE-Completeness of Inhabitation

We study the inhabitation problem for system $\lambda(-\cap\mathrm{I})$, formulated as follows:

Given an environment Γ and a type τ,
does there exist a λ-term M such that $\Gamma \vdash M : \tau$?

Our main result, Theorem 19 states that the problem is EXPSPACE-complete. We prove the exponential space upper bound in Section 3.1 and the lower bound in Section 3.2.

The upper bound is achieved by constructing an alternating exponential time algorithm. We exploit the fact, also noted in [7], that least upper bounds of types with respect to $\leq$ are definable. Our algorithm is quite different from the algorithm of [7], which relies on a fixpoint construction that reduces inhabitation to the emptiness problem for a class of context-free grammars. No specific upper bound was given in [7].

The lower bound is achieved by reduction from a slight variation of the *bus machine* model introduced in [17] which generically simulates alternating exponential time Turing machines. The reduction exploits distributivity properties of the subtyping relation together with the definability of least upper bounds.

3.1 EXPSPACE Upper Bound

Algorithm

Lemmas 9 and 11 together define an alternating algorithm to check type inhabitation in system $\lambda(-\cap \mathrm{I})$. The algorithm is formalized in Figure 2, which specifies an alternating Turing machine accepting on input Γ and τ if and only if there exists a normal inhabitant of τ in the environment Γ. (Note that, by Lemma 7, we only need to consider inhabitants in normal form.)

Recall ([3], see also, e.g., [10, section 16.2]) that the state set Q of an alternating Turing machine is partitioned into two subsets, $Q = Q_\exists \cup Q_\forall$. States in $Q_\exists$ are referred to as existential states, and states in $Q_\forall$ are referred to as universal states. A configuration whose state is in $Q_\forall$ is accepting if and only if all its successor configurations are accepting, and a configuration whose state is in $Q_\exists$ is accepting if and only if at least one of its successor configurations is accepting.

In the specification in Figure 2 we use shorthand notation for instruction sequences starting from existential states (CHOOSE ...) and instruction sequences starting from universal states (FORALL$(i = 1 \ldots k)\, S_i$). For example, a command of the form CHOOSE $x \in S$ branches from an existential state (associated with the CHOOSE-command) to successor states in which x gets assigned distinct elements of S. A command of the form FORALL$(i = 1 \ldots k)\, S_i$ branches from a universal state (associated with the FORALL-command) to successor states from which each instruction sequence S_i is executed.

For an organized type σ, we let $\|\sigma\|$ denote the maximal length of a path in σ. The symbol $\mathbb{P}_k(\sigma)$ denotes the set of all paths of length k or more in organized type σ.

Proposition 12. *Inhabitation in system* $\lambda(-\cap \mathrm{I})$ *is in* EXPSPACE.

Proof: We process problems of the form $\Gamma \vdash \tau$, where τ is organized and every $\Gamma(x)$ is a sum ($\bigoplus$) of organized types. There are two cases, nondeterministically chosen. One is justified by Lemma 9, the other by Lemma 11.

Case 1: For $\tau = \bigcap_{i\in I}(\tau_i \to \sigma_i)$, ask if $\Gamma,\ x : \bigoplus_{i\in I} \tau_i \ \vdash\ \bigcap_{i\in I} \sigma_i$.

Case 2: We guess x and k and a set P of paths in $\Gamma(x)$. That suffices to verify condition (2b) of Lemma 11.

```
        Input :  Γ, τ

1       G := {(Γ, τ)};

2       loop :
3       τ := organize(τ);
4       CHOOSE case ∈ {1, 2};
5       IF (case = 1) THEN
6          IF(τ = ⋂_{i∈I}(τ_i → σ_i)) THEN
7             IF(⨁_{i∈I} τ_i ∉ rn(Γ)) THEN
8                Γ := Γ ∪ {x : ⨁_{i∈I} τ_i} where x is fresh;
9             τ := ⋂_{i∈I} σ_i;
10            IF((Γ, τ) ∈ G) THEN REJECT
11            ELSE
12               G := G ∪ {(Γ, τ)};
13               GOTO loop;
14         ELSE REJECT;

15      IF (case = 2) THEN
16         CHOOSE (x : σ) ∈ Γ;
17         CHOOSE k ∈ {0, . . . , ‖σ‖};
18         CHOOSE P ⊆ ℙ_k(σ);
19         IF(⋂_{π∈P} tgt_k(π) ≤ τ) THEN
20            IF(k = 0) THEN ACCEPT
21            ELSE
22                  FORALL(i = 1 . . . k)
23                        τ := ⋂_{π∈P} arg_i(π);
24                        IF((Γ, τ) ∈ G) THEN REJECT
25                        ELSE
26                           G := G ∪ {(Γ, τ)};
27                           GOTO loop;
28         ELSE REJECT;
```

Fig. 2. Alternating EXPTIME-Turing machine

To handle condition (2a) assume for a moment that τ is a path. By Lemma 3, the inequality (2a) holds if and only if we have $tgt_k(\pi) \leq \tau$ for a single $\pi \in P$. If τ is an intersection of paths, we need one $\pi \in P$ for each.

At all stages, we ask questions of the form $\Gamma \vdash \tau$, where τ is an intersection of (organized form of) disjoint subterms of the original type. It takes linear space to write down each such τ. Types in Γ are sums of such disjoint subterms. The environment can only grow and one can optimize by assuming that no type is repeated (Figure 2 line 7). There are at most exponentially many such intersections and sums.

The variable G is an auxiliary device to prevent repeating goals. As inhabitation goals (Γ, τ) are generated (starting with the input goal), they are accumulated in G along branches of the computation tree to ensure that a goal (Γ, τ) is only considered in a branch if it has not already been considered in the branch.

It follows that the depth of alternating paths of computation are exponentially bounded and hence the (alternating) time used by the procedure is at most exponential. The proposition now follows from the well-known identity AEXPTIME = EXPSPACE [3, Corollary 3.6]. □

3.2 EXPSPACE Lower Bound

The lower bound is obtained by encoding the halting problem for bus machines into the inhabitation problem. A bus machine is an alternating computing device operating on a finite word (bus) of a fixed length. At every step the whole contents of the bus is updated according to one of the instructions of the machine. In addition new instructions may be created each time to be used in the future. A precise definition is as follows.

A *simple switch* over a finite alphabet $\mathcal{A}$ is a pair of elements of $\mathcal{A}$, written $a \leftarrow b$. A *labeled switch* is a quadruple, written $a \leftarrow b(c \leftarrow d)$, where the simple switch $c \leftarrow d$ is the *label*. Finally, a *universal switch* is a triple, written $a \leftarrow b \times c$.

Formally, a *bus machine* is a tuple $\mathcal{M} = \langle\, \mathcal{A}, m, w_0, w_1, \mathcal{I} \,\rangle$, where $\mathcal{A}$ is a finite alphabet, $m > 0$ is the *bus length* of $\mathcal{M}$ (the length of the words processed), w_0 and w_1 are words of length m over $\mathcal{A}$, called the *initial* and *final word*, respectively, and $\mathcal{I}$ is the set of *global instructions*.

Every global instruction is an m-tuple $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$ of sets of switches. Switches in I_i are meant to act on the i-th symbol of the bus. It is required that all switches in a given instruction $\mathbb{I}$ are of the same kind: either all are simple, or all are labeled, or all are universal. Therefore we classify instructions as simple, labeled, and universal. A *local instruction* is an m-tuple of simple switches and is considered a special case of a simple instruction (singletons at all coordinates).

A *configuration* of $\mathcal{M}$ is a pair $\langle\, w, \mathcal{J} \,\rangle$, where w is a word over $\mathcal{A}$ of length m, and $\mathcal{J}$ is a set of local instructions. The *initial* configuration is of course $\langle\, w_0, \varnothing \,\rangle$, and any configuration of the form $\langle\, w_1, \mathcal{J} \,\rangle$ is called *final*.

Suppose that $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$, and let $w = a_1 \ldots a_m$ and $w' = b_1 \ldots b_m$, $w'' = c_1 \ldots c_m$.

- If $\mathbb{I}$ is a simple instruction, and for every $i \leq m$ the simple switch $a_i \leftarrow b_i$ belongs to I_i, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J} \,\rangle$;
- If for every $i \leq m$ there is $a_i \leftarrow b_i(c_i \leftarrow d_i)$ in I_i, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J}' \,\rangle$, where $\mathcal{J}' = \mathcal{J} \cup \{\langle\, c_1 \leftarrow d_1, \ldots, c_m \leftarrow d_m \,\rangle\}$;
- If $\mathbb{I}$ is universal and $a_i \leftarrow b_i \times c_i$ is in I_i, for $i \leq m$, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{M} \langle\, w', \mathcal{J} \,\rangle$, and also $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{M} \langle\, w'', \mathcal{J} \,\rangle$.

A configuration $\langle\, w, \mathcal{J} \,\rangle$ is *accepting* iff it is either a final configuration, or

- There exists a non-universal instruction $\mathbb{I}$, such that $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J}' \,\rangle$ and $\langle\, w', \mathcal{J}' \,\rangle$ is accepting, or

– There is a universal instruction $\mathbb{I}$ such that we have $\langle w, \mathcal{J} \rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle w', \mathcal{J} \rangle$ and $\langle w, \mathcal{J} \rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle w'', \mathcal{J} \rangle$, where both $\langle w', \mathcal{J} \rangle$ and $\langle w'', \mathcal{J} \rangle$ are accepting.

The machine $\mathcal{M}$ *halts* iff the initial configuration is accepting.

Example 13. This example is repeated after [17], and is based on the ideas of [8]. Let $\mathcal{A} = \{0, 1\}$, $I = \{0 \leftarrow 0, 1 \leftarrow 1\}$, $I^+ = \{0 \leftarrow 1\}$, and $I^- = \{1 \leftarrow 0\}$. Take $\mathcal{M} = \langle \mathcal{A}, 4, 0000, 1111, \mathcal{I} \rangle$, where $\mathcal{I}$ consists of the following tuples (note that all global instructions are simple).

$$(I, I, I, I^+),\ (I, I, I^+, I^-),\ (I, I^+, I^-, I^-),\ (I^+, I^-, I^-, I^-).$$

The machine $\mathcal{M}$ makes $2^4 - 1$ steps and halts after it has seen all binary strings of length 4.

Example 14. This example demonstrates how local instructions work. We modify Example 13 so that $\mathcal{A} = \{0, 1, 2, 3\}$ and

$$I = \{0 \leftarrow 0(2{\leftarrow}2), 1 \leftarrow 1(3{\leftarrow}3)\},\quad I^+ = \{0 \leftarrow 1(2{\leftarrow}3)\},\quad I^- = \{1 \leftarrow 0(3{\leftarrow}2)\}.$$

In addition, let $I^* = \{1 \leftarrow 2\}$. Our new machine is $\mathcal{M} = \langle \mathcal{A}, 4, 0000, 3333, \mathcal{I} \rangle$, where $\mathcal{I}$ consists of the following tuples

$$(I, I, I, I^+),\ (I, I, I^+, I^-),\ (I, I^+, I^-, I^-),\ (I^+, I^-, I^-, I^-),\ (I^*, I^*, I^*, I^*).$$

For the first $2^4 - 1$ steps the machine behaves as in Example 13, using global instructions only. This time however, every application of a global instruction creates a new unique local instruction. After arriving at 1111 the machine rewrites the bus to 2222 and then executes one by one all the local instructions, finally reaching the final 3333. Observe that the number of local instructions is exponential and so is the (implicit) space needed to store them.

The following result was shown in [17].

Proposition 15. *The halting problem for bus machines is* EXPSPACE*-complete.*

The proof of Proposition 15 makes an essential use of both alternation and the ability of bus machines to create local instructions. Without the latter (when all global instructions are either simple or universal) the problem turns out to be complete in EXPTIME, and that is implicit in [8], because such bus machines are of the same power as alternating linear bounded automata.

Without alternation, as usual, the problems classify one step lower in the hierarchy (as EXPTIME- and PSPACE-complete, respectively).

We need a slightly improved version of Proposition 15. Call a bus machine $\mathcal{M} = \langle \mathcal{A}, m, w_0, w_1, \mathcal{I} \rangle$ *separated*, when the alphabet $\mathcal{A}$ is a union of m disjoint alphabets $\mathcal{A}_i$ and all links in the i-th component of every instruction are built exclusively from symbols in $\mathcal{A}_i$. We have

Corollary 16. *The halting problem for separated bus machines is* EXPSPACE*-complete.*

Proof: Simply replace $\mathcal{A}$ by a disjoint union of m copies of $\mathcal{A}$ and rewrite all switches and the initial and final words accordingly. The new machine halts if and only if the original one halts, because different components of the bus do not interact at all. □

Exponential Space Hardness

We reduce the halting problem for separated bus machines to the inhabitation problem. The principle of the encoding is similar to that in Section 3.3 of [17]. However, a major difference is that [17] deals with restricted types but unrestricted rules. In particular, rule ($\cap$I) makes it possible to simulate the m coordinates of the bus by using an intersection. To inhabit this intersection one looks for a single term solving the m problems represented by the components of the intersection.

Now, in system $\lambda(-\cap\mathrm{I})$, we have restricted rules. Without rule ($\cap$I) we cannot create a parallel problem. Instead we represent a bus as an intersection of atoms. By making the alphabets disjoint we ensure that an intersection of m atoms represents only one word of length m, because we know that the i-th symbol belongs to the i-th alphabet. Rather than simultaneously switching from a_i to b_i, at every $i = 1, \ldots, m$, we switch from $a_1 \cap \cdots \cap a_m$ to $b_1 \cap \cdots \cap b_m$. To make sure this works correctly, we need a new technique for coding rules, especially labeled rules. For this, a major instrument is Lemma 17 below.

So let $\mathcal{M} = \langle\, \mathcal{A}, m, w_0, w_1, \mathcal{I} \,\rangle$ be separated, and assume that the final word is $f_1 \ldots f_m$, and the initial word is $s_1 \ldots s_m$. Simple and universal switches of $\mathcal{M}$ are encoded by types of the form $a \to b$ and $a \to b \to c$. A labeled switch $a \leftarrow b(c \leftarrow d)$, with $a, b, c, d \in \mathcal{A}_i$, is encoded by

$$(((d \to c) \cap \textstyle\bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b) \to a.$$

The following property is useful.

Lemma 17. *Let $d_i, c_i \in \mathcal{A}_i$. Then*

$$\textstyle\bigoplus_{i=1}^{m} ((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) = \bigcap_{i=1}^{m} (d_i \to c_i).$$

Proof: According to the definition of $\bigoplus$, each path in the left-hand side must be of the form $d_i \cap \omega \cap \cdots \cap \omega \to c_i$, i.e., must be equal to $d_i \to c_i$. □

A local instruction $J = \langle\, c_1 \leftarrow b_1, \ldots, c_m \leftarrow b_m \,\rangle$ is encoded by the type $\alpha_J = \bigcap_{i=1}^{m} (b_i \to c_i)$. A global instruction $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$ is encoded as the intersection of types representing all switches in the union of $I_1, \ldots, I_n$. For example, a simple instruction $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$ with $I_j = \{a_k^j \leftarrow b_k^j \mid k = 1, \ldots, q_j\}$ for $j = 1, \ldots, m$, is represented by the type $\tau^{\mathbb{I}}$ defined by

$$\tau^{\mathbb{I}} = \bigcap \{b_k^j \to a_k^j \mid j = 1 \ldots m, k = 1 \ldots q_j\}.$$

Assuming $\mathcal{I} = \{\mathbb{I}_1, \ldots, \mathbb{I}_n\}$, we take $\Gamma = \{x_0 : f_1 \cap \cdots \cap f_m,\ x_1 : \tau^1, \ldots, x_n : \tau^n\}$, where each τ^j represents $\mathbb{I}_j$. In addition, let $\Gamma^{\mathcal{J}} = \Gamma \cup \{y_J : \alpha_J \mid J \in \mathcal{J}\}$, for any set $\mathcal{J}$ of local instructions.

A configuration $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle$ of the machine is now represented by the inhabitation problem $\Gamma^{\mathcal{J}} \vdash e_1 \cap \cdots \cap e_m$. Here is what we need to prove.

Lemma 18. *The configuration $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle$ is accepting if and only if there is a term M satisfying $\Gamma^{\mathcal{J}} \vdash M : e_1 \cap \cdots \cap e_m$.*

Proof: The proof of "only if" goes by induction with respect to the definition of acceptance.

Consider the case $\langle e_1 \dots e_m, \mathcal{J} \rangle \Rightarrow^{\mathbb{I}_j}_{\mathcal{M}} \langle b_1 \dots b_m, \mathcal{J} \rangle$, where $\mathbb{I}_j = \langle I_1, \dots, I_m \rangle$ is simple and where $e_i \leftarrow b_i$ is in the i-th component of $\mathbb{I}_j$, for $i = 1, \dots, m$, and $\langle b_1 \dots b_m, \mathcal{J} \rangle$ is accepting. By induction hypothesis, there exists M such that $\Gamma^{\mathcal{J}} \vdash M : b_1 \cap \dots \cap b_m$. By construction, we have $(x_j : \tau^j) \in \Gamma^{\mathcal{J}}$ with

$$\tau^j \leq \bigcap_{i=1}^m (b_i \to e_i) \leq \bigcap_{i=1}^m b_i \to \bigcap_{i=1}^m e_i$$

where the last inequality follows by distributivity properties of $\leq$. It follows that $\Gamma^{\mathcal{J}} \vdash x_j M : e_1 \cap \dots \cap e_m$. The similar case of a universal instruction is left out.

Consider the case of a labeled rule $\mathbb{I}_j$ corresponding to the variable x_j. We have a switch $e_i \leftarrow b_i(c_i \leftarrow d_i)$ at every component, and an accepting configuration $\langle b_1 \dots b_m, \mathcal{J} \cup \{J\} \rangle$, where $J = \langle c_1 \leftarrow d_1, \dots, c_m \leftarrow d_m \rangle$. By the induction hypothesis there is a term N such that

$$\Gamma^{\mathcal{J}}, y_J : \alpha_J \vdash N : b_1 \cap \dots \cap b_m.$$

By Lemma 17, we have $\alpha_J = \bigoplus_i ((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o))$, whence, by Lemma 9,

$$\Gamma^{\mathcal{J}} \vdash \lambda y_J\, N : \bigcap_i [((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b_i]$$

Since $\Gamma(x_j)$ contains all paths $(((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b_i) \to e_i$, we may conclude from Lemma 11 that $\Gamma^{\mathcal{J}} \vdash x_j(\lambda y_J\, N) : e_1 \cap \dots \cap e_m$.

The "if" part is shown by induction with respect to M, where M is assumed to be in normal form, cf. Lemma 7. Consider the case of $M = x_j M'$, where $(x_j : \tau^j) \in \Gamma^{\mathcal{J}}$ represents a simple instruction of the form $\mathbb{I}_j = \langle I_1, \dots, I_m \rangle$. Since $\Gamma^{\mathcal{J}} \vdash x_j M' : e_1 \cap \dots \cap e_m$, it follows from Lemma 5 that we must have $\tau^j \leq \zeta \to \bigcap_{i=1}^m e_i$, and $\Gamma^{\mathcal{J}} \vdash M' : \zeta$, for some ζ.

By Lemma 1, we have $\zeta \leq \bigcap \{b_t \mid t \in T\}$, and $\bigcap \{a_t \mid t \in T\} \leq \bigcap_{i=1}^m e_i$, where $\bigcap_{t \in T} (b_t \to a_t)$ is a component of τ^j. Since e_i and a_t are atoms, we actually have $\{e_i \mid 1 \leq i \leq m\} \subseteq \{a_t \mid t \in T\}$. Assume for simplicity that $\{1, \dots, m\} \subseteq T$, and $e_i = a_i$, for all i. So $\zeta \leq \bigcap \{b_i \mid i = 1 \dots m\}$ and thus $\Gamma^{\mathcal{J}} \vdash M' : b_1 \cap \dots \cap b_m$. By induction hypothesis, $\langle b_1 \dots b_m, \mathcal{J} \rangle$ is accepting. Moreover, by construction of τ^j together with $\tau^j \leq \zeta \to \bigcap_{i=1}^m e_i$, it must be that $e_i \leftarrow b_i$ is in the i-th component of $\mathbb{I}_j$, for $i = 1 \dots m$. It follows that $\langle e_1 \dots e_m, \mathcal{J} \rangle \Rightarrow^{\mathbb{I}_j}_{\mathcal{M}} \langle b_1 \dots b_m, \mathcal{J} \rangle$, and hence $\langle e_1 \dots e_m, \mathcal{J} \rangle$ is accepting.

The case of a universal instruction is similar. Again the most complex case is when $M = x(\lambda y\, N)$. If $\Gamma^{\mathcal{J}} \vdash M : e_1 \cap \dots \cap e_m$ then, for every i, there must be a component $(((a_i \to b_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to c_i) \to e_i$ in $\Gamma(x)$ such that

$$\Gamma^{\mathcal{J}} \vdash \lambda y\, N : \bigcap_i (((a_i \to b_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to c_i).$$

Using again Lemmas 11 and 17, we see that $\Gamma^{\mathcal{J}}, y : \alpha_J \vdash \bigcap_i c_i$, for an appropriate J, and here we apply the induction hypothesis. □

Theorem 19. *The inhabitation problem for* $\lambda(-\cap \mathrm{I})$ *is* EXPSPACE*-complete.*

Proof: The problem is in EXPSPACE by Proposition 12, hardness follows from Corollary 16. Indeed, by Lemma 18, the machine $\mathcal{M}$ halts if and only if type

$\gamma_1 \to \cdots \to \gamma_k \to s_1 \cap \cdots \cap s_m$ is inhabited, where γ_i are all types from the environment Γ. (Note that the initial $\mathcal{J}$ is empty.) □

4 PSPACE-Completeness without Subtyping

Systems of intersection types are often defined without subtyping, but with the intersection elimination rule ($\cap$E) (which is derivable in presence of the subsumption rule). Such a restriction of $\lambda(-\cap\mathrm{I})$ is given in Figure 3.

$$\frac{}{\Gamma, x:\tau \vdash x:\tau}(\mathrm{var}) \qquad \frac{\Gamma \vdash M:\tau\to\tau' \quad \Gamma \vdash N:\tau}{\Gamma \vdash (M\ N):\tau'}(\to\mathrm{E})$$

$$\frac{\Gamma, x:\tau \vdash M:\tau'}{\Gamma \vdash \lambda x\, M:\tau\to\tau'}(\to\mathrm{I}) \qquad \frac{\Gamma \vdash M:\tau_1\cap\tau_2}{\Gamma \vdash M:\tau_i}(\cap\mathrm{E})$$

Fig. 3. Type system $\lambda(-\cap\mathrm{I})$ without subtyping

Define a relation $\sqsubseteq$ as the least quasi-order such that:

- $\tau \cap \sigma \sqsubseteq \tau$;
- If $\sigma \sqsubseteq \sigma'$ then $\tau \to \sigma \sqsubseteq \tau \to \sigma'$.

Lemma 20. *If $\Gamma \vdash xN_1 \ldots N_k : \sigma$ then $\Gamma(x) \sqsubseteq \tau_1 \to \cdots \to \tau_k \to \sigma$, where $\Gamma \vdash N_i : \tau_i$, for all i.*

Proof: Induction. □

A derivation is *long normal* when it only uses the following rules:

$$\frac{\Gamma(x:\tau) \vdash N:\sigma}{\Gamma \vdash \lambda x\, N:\tau\to\sigma} \qquad \frac{\Gamma \vdash N_i:\tau_i \quad (i=1,\ldots,k)}{\Gamma \vdash xN_1\ldots N_k:\alpha}\,(\Gamma(x) \sqsubseteq \tau_1 \to\cdots\to \tau_k \to \alpha)$$

where α is not an arrow type. The next lemma is almost literally the same as proving that an inhabited simple type has a long normal inhabitant.

Lemma 21. *If $\Gamma \vdash M : \tau$ then there is a long normal derivation of $\Gamma \vdash M' : \tau$, for some M'.*

Proof: Without loss of generality, we can take M normal. We use induction with respect to the size of M. If $\tau = \sigma \to \rho$ and $M = \lambda z\, N$, apply induction to N. If $M = zN_1 \ldots N_k$, write τ as $\tau = \rho_1 \to \cdots \to \rho_m \to \alpha$, with α not an arrow type. Use Lemma 20, and apply induction to $\Gamma \vdash N_i : \tau_i$ to obtain terms N_i' such that $\Gamma \vdash N_i' : \tau_i$ have long normal derivations. Finally define M' as the term $\lambda y_1 \ldots y_m.\, zN_1' \ldots N_k' y_1 \ldots y_m$. □

Corollary 22. *Inhabitation in system* $\lambda(-\cap\mathrm{I})$ *without subtyping is* PSPACE*-complete.*

Proof: By Lemma 21 an inhabited type has an inhabitant with a long normal derivation. Therefore, a PSPACE inhabitation algorithm can be obtained as a slight modification of the ordinary Wajsberg/Ben-Yelles procedure for simple types, as in e.g. [15]. The only difference is that, when looking for a proof of a non-arrow type α, one chooses a "path" of the form $\tau_1 \to \cdots \to \tau_k \to \alpha$ where $\Gamma(x) \sqsubseteq \tau_1 \to \cdots \to \tau_k \to \alpha$ (rather than $\Gamma(x) = \tau_1 \to \cdots \to \tau_k \to \alpha$), for some x. A path is selected nondeterministically by inspecting the tree of $\Gamma(x)$ from the root towards the leaves (may stop at an internal $\cap$-node). The case $\Gamma \vdash \tau \to \sigma$ is reduced as usual to $\Gamma \vdash \sigma$ or $\Gamma, x : \tau \vdash \sigma$, with fresh x, depending on whether $\tau \in rn(\Gamma)$ or not.

The PSPACE-hardness follows easily from PSPACE-hardness of inhabitation in simple types, since $\lambda(-\cap\mathrm{I})$ without subtyping is easily seen to be a conservative extension of the simple typed system $\lambda_\to$ (if Γ contains only simple types, then $\Gamma \vdash M : \tau$ in $\lambda(-\cap\mathrm{I})$ is equivalent to $\Gamma \vdash M : \tau$ in $\lambda_\to$). □

5 Conclusion

We have considered the question of complexity of inhabitation for a natural fragment of the intersection typed λ-calculus, in which intersections are *explicit*, in the sense of "Church-style" type assumptions or, equivalently, in the sense that intersections cannot be introduced. Decidability was shown by Kurata and Takahashi [7], but the question of complexity was left open. In the present paper we have settled the question: the problem is EXPSPACE-complete. We have emphasized the importance of subtyping by pointing out that the problem is PSPACE-complete in the absence of subtyping. Since decidable inhabitation can be seen as a basis for automatic program synthesis problems [12], our result determines the complexity of synthesis for the "Church-style fragment" of the intersection typed λ-calculus. More generally, our result adds to the emerging systematic understanding of the fine structure of intersection type systems.

References

1. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. Journal of Symbolic Logic 48(4), 931–940 (1983)
2. Barendregt, H., Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., Barendregt, H.P.: Lambda calculi with types. In: Handbook of Logic in Computer Science, pp. 117–309. Oxford University Press (1992)
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM 28(1), 114–133 (1981)
4. Coppo, M., Dezani-Ciancaglini, M.: An extension of basic functionality theory for lambda-calculus. Notre Dame Journal of Formal Logic 21, 685–693 (1980)
5. Dezani-Ciancaglini, M., Hindley, R.: Intersection types for combinatory logic. Theoretical Computer Science 100(2), 303–324 (1992)

6. Hindley, J.R.: The Simple Semantics for Coppo-Dezani-Sallé Types. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming 1982. LNCS, vol. 137, pp. 212–226. Springer, Heidelberg (1982)
7. Kurata, T., Takahashi, M.: Decidable Properties of Intersection Type Systems. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902, pp. 297–311. Springer, Heidelberg (1995)
8. Kuśmierek, D.: The Inhabitation Problem for Rank Two Intersection Types. In: Ronchi Della Rocca, S. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 240–254. Springer, Heidelberg (2007)
9. Leivant, D.: Polymorphic type inference. In: Proc. 10th ACM Symp. on Principles of Programming Languages, pp. 88–98. ACM (January 1983)
10. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
11. Pottinger, G.: A type assignment for the strongly normalizable lambda-terms. In: Hindley, J., Seldin, J. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 561–577. Academic Press (1980)
12. Rehof, J., Urzyczyn, P.: Finite Combinatory Logic with Intersection Types. In: Ong, L. (ed.) TLCA 2011. LNCS, vol. 6690, pp. 169–183. Springer, Heidelberg (2011)
13. Salvati, S.: Recognizability in the Simply Typed Lambda-Calculus. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS, vol. 5514, pp. 48–60. Springer, Heidelberg (2009)
14. Statman, R.: Intuitionistic propositional logic is polynomial-space complete. Theoretical Computer Science 9, 67–72 (1979)
15. Urzyczyn, P.: Inhabitation in Typed Lambda-Calculi (a Syntactic Approach). In: de Groote, P., Hindley, J.R. (eds.) TLCA 1997. LNCS, vol. 1210, pp. 373–389. Springer, Heidelberg (1997)
16. Urzyczyn, P.: The emptiness problem for intersection types. Journal of Symbolic Logic 64(3), 1195–1215 (1999)
17. Urzyczyn, P.: Inhabitation of Low-Rank Intersection Types. In: Curien, P.-L. (ed.) TLCA 2009. LNCS, vol. 5608, pp. 356–370. Springer, Heidelberg (2009)
18. Urzyczyn, P.: The logic of persistent intersection. Fundamenta Informaticae 103, 303–322 (2010)

On State Sequences Defined by Reaction Systems

Arto Salomaa

Turku Centre for Computer Science
Joukahaisenkatu 3–5 B, 20520 Turku, Finland
asalomaa@utu.fi

Abstract. The paper investigates sequences generated by *reaction systems*. Arbitrary sequences can be generated if the cardinalities of the sets of *reactants* and *inhibitors* are unbounded. Most of the paper investigates systems where both of these cardinalities equal 1. A general result is obtained concerning sequences generated by systems with *interaction*. New estimates are obtained for lengths of sequences in the non-interactive case.

Keywords: reaction system, inhibitor, reactant, state sequence, interactive process.

1 Introduction and Basic Definitions

Reaction systems were introduced by A. Ehrenfeucht and G. Rozenberg as a formal model of interactions between biochemical reactions. The reader is referred to [3] for some of the original motivation and initial setup. Each reaction is characterized by its set of *reactants*, each of which has to be present for the reaction to take place, by its set of *inhibitors*, none of which is allowed to be present, and by its set of *products*, each of which will be present after a successful reaction. Thus, a single reaction is based on facilitation and inhibition.

The model of reaction systems turned out to be suitable in a large variety of different setups, and the possibilities are by far not exhausted. The reader is referred to [1] for a survey.

However, in this paper we focus on some formal properties of the basic model, neglecting eventual applications. The paper is largely self-contained. We will now define the notions discussed in the paper.

Definition 1. *A* reaction *over the* base set S *is a triple*

$$\rho = (R, I, P),$$

where R, I *and* P *are nonempty subsets of* S *such that* R *and* I *do not intersect. The three sets are referred as* reactants, inhibitors *and* products*, respectively.*

Observe that no specific assumptions are made about the set P. In particular, it may contain elements of $R \cup I$. In this paper S will always denote the base set. We usually denote its elements by natural numbers.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 271–282, 2012.

The *cardinality* of a finite set X is denoted by $\sharp X$. The *empty set* is denoted by ϕ.

Definition 2. *A* reaction system $\mathcal{A}_S$ *over the base set* S *is a finite nonempty set of reactions*

$$\{\rho_i |\ 1 \leq i \leq k\},$$

over S.

We will omit the index S from $\mathcal{A}_S$ whenever S is understood. We now come to the basic definitions, dealing with operations and sequences.

Definition 3. *Consider a reaction* $\rho = (R, I, P)$ *over* S *and a subset* T *of* S. *The set* T *is* enabled *(with respect to* ρ*), in symbols* $en_\rho(T)$, *if* $R \subseteq T$ *and* $I \cap T = \phi$. *If* T *is (resp. is not) enabled, then we define the* result *by*

$$res_\rho(T) = P \text{ (resp. } = \phi).$$

For a reaction system $\mathcal{A} = \{\rho_i |\ 1 \leq i \leq k\}$, *we define the result by*

$$res_{\mathcal{A}}(T) = \bigcup_{i=1}^{k} res_{\rho_i}(T).$$

As an example, let $\mathcal{A}_1$ be a reaction system over the base set $\{1, 2, 3\}$, consisting of the three reactions

$$\rho_1 = (\{1, 2\}, \{3\}, \{3\}),\ \rho_2 = (\{1\}, \{3\}, \{1\}),\ \rho_3 = (\{2\}, \{1\}, \{1, 2\}).$$

Consider $T = \{1, 2\}$. Then $en_{\rho_1}(T)$ and $en_{\rho_2}(T)$, whereas $en_{\rho_3}(T)$ does not hold. Consequently,

$$res_{\rho_1}(T) = \{3\},\ res_{\rho_2}(T) = \{1\},\ res_{\rho_3}(T) = \phi,\ res_{\mathcal{A}_1}(T) = \{1, 3\}.$$

Definition 3 and our example exhibit an important feature of reaction systems. Whenever an element is in a set, it is considered to be there always when needed. Thus, the element 1 of T is not "consumed" in the application of the reaction ρ_1 but is also available for ρ_2 when $res_{\mathcal{A}_1}(T)$ is computed. In this sense there is no "conflict" between ρ_1 and ρ_2. This feature makes reaction systems different from many other models, for instance, Petri nets.

For any $\mathcal{A}$ and T, the result $res_{\mathcal{A}}(T)$ is always a unique subset T' of S. If $res_{\mathcal{A}}(T) = T'$, we use the notation

$$T \Rightarrow_{\mathcal{A}} T',$$

or simply $T \Rightarrow T'$ if $\mathcal{A}$ is understood. If

$$res_{\mathcal{A}}(T_i) = T_{i+1},\ 0 \leq i \leq m - 1,$$

we write

$$T_0 \Rightarrow T_1 \Rightarrow \ldots \Rightarrow T_m$$

and call $T_0, T_1, \ldots, T_m$ a *sequence* of *length* m generated (or defined) by the reaction system $\mathcal{A}$. Sometimes the sets T_i are referred to as *states* and the sequence itself as a *state sequence*.

Since $res_{\mathcal{A}}(T)$ is uniquely determined by T, and since there are only $2^{\sharp S}$ subsets of S, one of the following two alternatives always occurs, for large enough m, for sequences

$$T_0 \Rightarrow T_1 \Rightarrow \ldots \Rightarrow T_m.$$

1. $T_m = \phi$. Then we say that the sequence is a *terminating sequence* of length m. In this case $en_\rho(T_{m-1})$ holds for no reaction ρ in $\mathcal{A}$.
2. $T_m = T_{m_1}$, for some $m_1 < m$. In this case we say that the sequence has (or ends with) a *cycle* of length $m - m_1$.

As another example, let $\mathcal{A}_2$ be a reaction system over the base set $\{1, 2, 3\}$, consisting of the four reactions

$$\rho_1 = (\{1\}, \{2\}, \{1\}),\ \rho_2 = (\{1\}, \{3\}, \{2\}),$$
$$\rho_3 = (\{2\}, \{3\}, \{3\}),\ \rho_4 = (\{3\}, \{1\}, \{1, 3\}).$$

We obtain now a cycle

$$\{1\} \Rightarrow \{1, 2\} \Rightarrow \{2, 3\} \Rightarrow \{1, 3\} \Rightarrow \{1\}$$

of length 4, as well as another cycle

$$\{2\} \Rightarrow \{3\} \Rightarrow \{1, 3\} \Rightarrow \{1\} \Rightarrow \{1, 2\} \Rightarrow \{2, 3\} \Rightarrow \{1, 3\},$$

also of length 4. In the latter case we also have an initial part of length 2. The whole sequence contains all proper nonempty subsets of the base set. This shows that $\{1, 2, 3\} \Rightarrow \phi$ is the only terminating sequence (starting with a nonempty set).

Reaction systems are classified according to the maximal cardinalities of the sets of reactants and inhibitors.

Definition 4. *A reaction system $\mathcal{A}$ is a* (k, l) system *if the conditions $\sharp R \leq k$ and $\sharp I \leq l$ are satisfied for every reaction (R, I, P) in $\mathcal{A}$.*

The example $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) considered above is a $(2, 1)$ (resp. $(1, 1)$) system.

We will consider below in Section 3 *interactive processes*. This means that the sequence of a reaction system is combined with an *auxiliary sequence*. In this way most of the restrictions concerning sequences, resulting from the definitions given above, are removed and the sequences may continue indefinitely.

Definition 5. *Consider a reaction system $\mathcal{A}$ over the base set S, as well as an* auxiliary sequence

$$\mathcal{C} : C_0, C_1, C_2, \ldots,\ C_i \subseteq S,\ i \geq 0.$$

A sequence $T_0, T_1, T_2, \ldots$, of subsets of S is generated *(or defined) by the* interactive process $(\mathcal{A}, \mathcal{C})$ *if*

$$T_{i+1} = res_{\mathcal{A}}(T_i \cup C_i),\ \text{for all } i \geq 0.$$

We use the notation

$$T_0 \Rightarrow_{C_0} T_1 \Rightarrow_{C_1} T_2 \Rightarrow_{C_2} \ldots$$

We consider only finite prefixes of the auxiliary sequence $\mathcal{C}$. If each of the sets C_i is empty, we obtain a sequence of the reaction system $\mathcal{A}$, as defined above. In Definition 5 the sets C_i appear only in $res_{\mathcal{A}}(T_i \cup C_i)$. Definitions customary in the literature consider sets $T_i \cup C_i$. The difference is unessential but the notation in Section 3 becomes more complicated if sets $T_i \cup C_i$ are used.

A brief outline of the contents of the main part of the paper follows. In Section 2 we introduce a special type of reactions, the *maximally inhibited* ones, and show that they generate any prechosen sequence. In this case, the cardinalities of the sets R and I are unbounded. The rest of the paper deals with $(1,1)$ systems. Even then any sequence can be generated as a subsequence of a system using a simple auxiliary sequence. This will be shown in Section 3, whereas Section 4 improves the previously obtained lower bound for the maximal length of sequences generated by $(1,1)$ systems without an auxiliary sequence.

2 Maximally Inhibited Reactions

In spite of the simplicity of the basic definitions, reaction systems can exhibit quite complicated behavior. In particular, the terminating state sequences, as well as the cycles, can be very long. This will be very clearly visible if one considers *maximally inhibited reactions and reaction systems.*

Definition 6. *A reaction over the base set S is* maximally inhibited *if it is of the form $(R, S - R, P)$. A reaction system is* maximally inhibited *if every one of its reactions is maximally inhibited.*

A reaction being maximally inhibited imposes severe restrictions on states being enabled. On the other hand, one may construct terminating state sequences, as well as cycles, of maximal length. One may even have the states in any preassigned order.

We begin with an example. Consider the base set $S = \{1,2,3,4\}$. We want to construct a reaction system such that all nonempty subsets of S appear in its terminating state sequence in the order

$$\{1,3,4\},\{2,3\},\{4\},\{2,3,4\},\{1,2\},\{1,2,4\},\{3\},\{1\},$$
$$\{1,3\},\{1,2,3\},\{2,4\},\{3,4\},\{1,4\},\{2\},\{1,2,3,4\}.$$

Observe that the whole set S can appear in a state sequence only at the end. This follows from the fact that the set of inhibitors is, by definition, nonempty. By the same reason, S can never appear in a cycle.

The reaction system defined below has the above sequence as its (terminating) state sequence, provided $\{1,3,4\}$ is the first state. The system is maximally inhibited and consists of the following reactions:

$$(\{1\},\{2,3,4\},\{1,3\}),(\{2\},\{1,3,4\},\{1,2,3,4\}),(\{3\},\{1,2,4\},\{1\}),$$
$$(\{4\},\{1,2,3\},\{2,3,4\}),(\{1,2\},\{3,4\},\{1,2,4\}),(\{1,3\},\{2,4\},\{1,2,3\}),$$
$$(\{1,4\},\{2,3\},\{2\}),(\{2,3\},\{1,4\},\{4\}),(\{2,4\},\{1,3\},\{3,4\}),$$
$$(\{3,4\},\{1,2\},\{1,4\}),(\{1,2,3\},\{4\},\{2,4\}),(\{1,2,4\},\{3\},\{3\}),$$
$$(\{1,3,4\},\{2\},\{2,3\}),(\{2,3,4\},\{1\},\{1,2\}).$$

At each state in the sequence, exactly one reaction is applicable. This is generally true for maximally inhibited reaction systems.

Lemma 1. *In a maximally inhibited reaction system, for any $T \subseteq S$, only reactions whose set of reactants equals T can be applied.*

Proof. Any other possible reaction has at least one inhibitor contained in T. □

We now generalize the above example. Consider an arbitrary enumeration

$$T_1, T_2, \ldots, T_{2^n-1}$$

of all nonempty subsets of the set S with n elements, ending with the set S itself. Consider, further, the (maximally inhibited) reaction system with the base set S and reactions

$$(T_i, S - T_i, T_{i+1}),\ 1 \leq i \leq 2^n - 2.$$

Then, starting with T_1, the reaction system generates the terminating sequence given above. If we change the reaction $(T_{2^n-2}, S - T_{2^n-2}, T_{2^n-1})$ to the reaction $(T_{2^n-2}, S - T_{2^n-2}, T_1)$, we get a cycle of length $2^n - 2$. Hence, we obtain the following result.

Theorem 1. *Given the base set S with n elements, there exists effectively a reaction system with a terminating state sequence of length $2^n - 1$, as well as a reaction system with a cycle of length $2^n - 2$. Moreover, the reaction systems can be constructed in such a way that the elements in the terminating state sequence and cycle are in any preassigned order.*

3 Power of Interaction: A Construction Using Nondeterministic Automata

As we have seen, functional constructions based on reaction systems are easy if the cardinalities of the sets R and I are not bounded. From now on we consider $(1,1)$ reaction systems, that is, both of the sets are singletons. This case is very natural from the point of view of combinatorics. In some sense it is also related to context-free rewriting.

Now *interaction* provides a powerful tool. We will obtain below a result resembling Theorem 1 for $(1,1)$ reaction systems, provided one of two specific symbols is added to the state of the system at each step of the computation. The constructions resemble the ones used for nondeterministic automata, [5,6].

Remark. In what follows, we apply the following *notational simplification* if there is no danger of confusion. We identify elements with singleton sets and

write x instead of $\{x\}$. We often deal with sets such as $\{2,3,5,7\}$. If there is no danger of confusion, we replace this notation by 2357. Thus, if we speak of the reaction $(1,2,235)$, this means $(\{1\},\{2\},\{2,3,5\})$ when the notational simplification is not used.

As an example, consider the reaction system with the base set $\{1,2,3,4,p,q\}$ and reactions

$$(1,p,4),\ (2,p,1),\ (3,p,2),\ (4,p,3),$$
$$(1,q,13),\ (2,q,2),\ (3,q,4).$$

By interaction with a sequence consisting of p's and q's, we obtain the following state sequence:

$$\begin{aligned} 34 \Rightarrow_q 23 \Rightarrow_q 12 \Rightarrow_q 14 \Rightarrow_p 13 \Rightarrow_q 24 \Rightarrow_p 2 \Rightarrow_q 1 \Rightarrow_q 4 \\ \Rightarrow_q 3 \Rightarrow_q 2 \Rightarrow_q 1 \Rightarrow_p 13 \Rightarrow_p 134 \Rightarrow_q 234 \Rightarrow_q 123 \\ \Rightarrow_q 124 \Rightarrow_q 134 \Rightarrow_q 234 \Rightarrow_q 123 \Rightarrow_p 1234 \Rightarrow_{\{p,q\}} \phi. \end{aligned}$$

Observe that every subset of the set $\{1,2,3,4\}$ appears in this sequence. Some appear seemingly with repetitions but in such cases the continuation of the auxiliary sequence of p's and q's will be different. We now generalize this example, using ideas from automata theory. Indeed, p and q can be viewed as input letters of a nondeterministic finite automaton.

For $n \geq 4$, we define a $(1,1)$ reaction system $\mathcal{A}_n$ as follows. The base set consists of $n+2$ elements:

$$S = \{1,2,\ldots,n,p,q\}.$$

(The restriction $n \geq 4$ is unessential. Our results hold also for smaller values of n, with a slight modification in the construction. The details are left to the reader. Also [6] can be consulted.) The set of reactions of $\mathcal{A}_n$ consists of the following reactions:

$$(2,q,1),\ (3,q,2),\ (1,q,3),\ (i,q,i),\ i>3;$$
$$(n,p,1),\ (n,p,2),\ (i,p,i+1),\ 2 \leq i \leq n-1.$$

We now consider interactive processes, where the auxiliary sequence is a sequence of p's and q's. By *states* we mean subsets of the set $S_1 = \{1,2,\ldots,n\}$. A state Y is *reachable* from a state X, in symbols $X \Rightarrow^* Y$, if $X \Rightarrow_p Y$ or $X \Rightarrow_q Y$, or else, Y is reachable from a set Z which is reachable from X. Moreover, X is reachable from itself. We sometimes use also the self-explanatory notations $X \Rightarrow_p^* Y$ or $X \Rightarrow_q^* Y$.

It is immediate that from the empty set and the whole set S_1 only the sets themselves are reachable. We will see below that this is the only exception. For this purpose, we will prove two lemmas.

Lemma 2. *Every subset of S_1 is reachable from the set $\{1\}$.*

Proof. Observe first the following reachability chain:

$$1 \Rightarrow_p^* 2 \Rightarrow_q 3 \Rightarrow_q \ldots \Rightarrow_q n \Rightarrow_q \{1,2\} \Rightarrow_q 3 \Rightarrow_p^* 1.$$

Also $1 \Rightarrow_q \phi$.

To prove the lemma, we assume inductively that every subset of S_1 with cardinality $k-1,\ 2 \leq k \leq n$, is reachable. Consider an arbitrary subset X of S_1,

$$X = \{x_1, x_2, \ldots, x_k\},\ x_1 < x_2 < \ldots < x_k,$$

with cardinality k. To show the reachability of X, we distinguish several cases.

Assume that $x_1 = 1,\ x_2 = 2$. Then the set $X_1 = \{x_3 - 1, \ldots, x_k - 1, n\}$ is of cardinality $k-1$, and $X_1 \Rightarrow_q X$.

Assume next that $x_1 = 1,\ x_2 = 3$. Then $x_3 \geq 4$ and the set $X_2 = \{1, 2, x_3, \ldots, x_k\}$ is reachable by the preceding paragraph. But $X_2 \Rightarrow_p X$ and, hence, X is reachable also in this case.

If $x_1 = 2,\ x_2 = 3$, the relation $X_2 \Rightarrow_p^* X$ shows the reachability of X.

The following three possibilities remain.

Case A. $x_1 = 1,\ x_2 \geq 4$.

Case B. $x_1 = 2,\ x_2 \geq 4$.

Case C. $x_1 \geq 3$.

We now introduce a total order $<_{ksub}$ for subsets of S_1 with cardinality k. Consider two such subsets

$$Y = \{y_1, y_2, \ldots, y_k\},\ Z = \{z_1, z_2, \ldots, z_k\},$$

where the elements are in increasing order. Then $Y <_{ksub} Z$ if $y_k - y_1 < z_k - z_1$, or else

$$y_k - y_1 = z_k - z_1,\ y_1 = z_1, \ldots, y_{j-1} = z_{j-1}, y_j < z_j,$$

for some $j,\ 1 \leq j \leq k$. (Thus, we first compare the differences between the last and first elements and then, in case of equality between the differences, compare the first elements different in the two sets.) We have shown that the set $\{1, 2, 3, \ldots, k\}$, the first in this order, is reachable.

Now another induction is applied. We assume inductively that all sets $Y,\ Y <_{ksub} X$, are reachable.

In Case A we choose $Y_1 = \{2, x_2, \ldots, x_k\}$. Then $Y_1 <_{ksub} X$ because $x_k - 2 < x_k - 1$. On the other hand, $Y_1 \Rightarrow_p X$.

In Case B we choose $Y_2 = \{3, x_2, \ldots, x_k\}$. Then $Y_2 <_{ksub} X$ because $x_k - 3 < x_k - 2$. Also now we get $Y_2 \Rightarrow_p X$.

In Case C we choose $Y_3 = \{x_1 - 1, x_2 - 1, \ldots, x_k - 1\}$. Then $Y_3 <_{ksub} X$ because $x_1 - 1 < x_1$. But $Y_3 \Rightarrow_q X$.

Consequently, we have established the reachability of X in all cases and, thus, completed the induction. □

The next lemma is a reverse of Lemma 2.

Lemma 3. *The set* $\{1\}$ *is reachable from every proper nonempty subset of* S_1.

Proof. We apply again an inductive argument. By the reachability chain at the beginning of the proof of Lemma 2, it suffices to show that, from any

$$X \subseteq S_1,\ 2 \leq \sharp X = k \leq n-1,$$

a subset of S_1 with cardinality $k-1$ is reachable. We write X in the form

$$X = X_1 \cup X_2,\ X_1 \subseteq \{1,2,3\},\ X_2 \subseteq \{4,\ldots,n\}.$$

and distinguish several cases, depending on the cardinality of X_1. The relation $\Rightarrow_p$ does not affect the set X_2, whereas it changes singletons in $\{1,2,3\}$, as well as two-element subsets of $\{1,2,3\}$ arbitrarily. Therefore, we may assume that, whenever $\sharp X_1 = 1$ (resp. $\sharp X_1 = 2$), then $X_1 = \{1\}$ (resp. $X_1 = \{1,2\}$).

Case 1. $\sharp X_1 = 3$. Consequently, $0 \leq \sharp X_2 < n-3$. If n is not in X_2, then an application of $\Rightarrow_q$ gives a subset with cardinality $k-1$. If $n \in X_2$, then $X \Rightarrow_q X_1 \cup X_2^1$, where X_2^1 is obtained from X_2 by removing n, adding 4, and replacing the other elements x, $4 \leq x < n$, with $x+1$. Thus, the cardinalities of X_1 and X_2 remain unchanged. Since $\sharp X_2 < n-3$, by repeated applications of $\Rightarrow_q$ we eventually reach a set not containing n, from which another application of $\Rightarrow_q$ produces a set of cardinality $k-1$. The end result in this case is always

$$Y_1 \cup Y_2,\ 1 = \sharp Y_1,\ Y_1 \subseteq \{1,2,3\},\ \sharp X_2 + 1 = \sharp Y_2,\ Y_2 \subseteq \{4,\ldots,n\}.$$

Case 2. $X_1 = \{1,2\}$. If $\sharp X_2 = n-3$, then $n \in X_2$, and an application of $\Rightarrow_q$ brings us back to Case 1. If $\sharp X_2 < n-3$, we apply again $\Rightarrow_q$. If n is not in X_2, we reach the set $\{3\} \cup X_2^1$ of cardinality $k-1$. If $n \in X_2$, we reach a set of the same cardinality as X, and are back in Case 1.

Case 3. $X_1 = \{1\}$. When we now apply $\Rightarrow_q$, we either get a set of cardinality $k-1$, or are back in Case 2.

Case 4. $X_1 = \phi$. In this case we first increase the cardinality, after which we can decrease it twice. We know that $2 \leq \sharp X_2 \leq n-3$. By successive applications of $\Rightarrow_q$, we eventually reach the set

$$\{1,2\} \cup X_2^2,\ \sharp X_2^2 = \sharp X_2 - 1.$$

If $n \in X_2^2$, an application of $\Rightarrow_q$ yields the set $\{1,2,3\} \cup X_2^3$, where $\sharp X_2^3 = \sharp X_2 - 2$. We now reduce the cardinality to k, as in Case 1, after which a further reduction takes place as in Case 3. (Recall the form of the end result in Case 1.) If n is not in X_2^2, then an application of $\Rightarrow_q$ produces the set $\{3\} \cup X_2^4$ with cardinality k, and after applying $\Rightarrow_p$, we are back in Case 3. Thus, we have completed the induction in all cases. □

Since Case 4 is the trickiest, we still illustrate the procedure. We have $n = 9$ in our example and start with the set $\{5,6,7,8,9\}$.

$$\begin{aligned} 56789 &\Rightarrow_q 126789 \Rightarrow_q^* 123459 \\ &\Rightarrow_q 123456 \Rightarrow_q 34567 \Rightarrow_p^* 14567 \Rightarrow_q 5678 \end{aligned}$$

Consider now interactive state sequences of the reaction system $\mathcal{A}_n$, where the auxiliary sequence is a sequence of p's and q's. By Lemmas 2 and 3, we now obtain immediately the following result.

Theorem 2. *Consider the reaction system $\mathcal{A}_n$ as defined above. Consider, further, an arbitrary sequence T_i, $1 \leq i \leq 2^n - 2$, of proper nonempty subsets of the set S_1. There is a terminating state sequence, as well as a cycle, of $\mathcal{A}_n$, where the sequence T_i appears as a subsequence.*

Observe that the sequence T_i appears as a *subsequence* of the sequence of $\mathcal{A}_n$. The latter sequence contains "junk" not in T_i.

4 Long Terminating State Sequences and Cycles in (1, 1) Reaction Systems

We will now discuss the basic variant, $(1,1)$ reaction systems without interaction. It was shown in [2] that in this case there is a terminating state sequence of length $3 \cdot 2^k - 3$, provided the base set is of cardinality $3k$. Moreover, there is a cycle of length $3 \cdot 2^k - 1$, provided the base set is of cardinality $3k + 3$. We now improve these results.

The results in [2] were obtained by recursion. Immediate improvements result if the *basis* of recursion is changed. Indeed, the reaction system with the base set $\{1, 2, 3\}$ and reactions

$$(1, 2, 2),\ (2, 1, 3),\ (3, 2, 13)$$

has the terminating state sequence

$$1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 13 \Rightarrow 123 \Rightarrow \phi$$

of length 5. (This is to be contrasted with the length 3 used in [2].) On the other hand, the following result was established in [2].

Lemma 4. *Assume that a $(1,1)$ reaction system $\mathcal{A}$ has a terminating state sequence of length $m \geq 1$. Adding three elements to the base set, one can construct another $(1,1)$ reaction system having a terminating state sequence of length $2m + 3$. Moreover, one can construct a further $(1,1)$ reaction system, again adding three elements to the base set of $\mathcal{A}$, with a cycle of length $m + 2$.*

Our example above shows that, for $k = 1$, there is a $(1,1)$ reaction system with $\sharp S = 3k$ and with a terminating state sequence of length 5. According to Lemma 4, whenever there is a $(1,1)$ reaction system with $\sharp S = 3k$ and with a terminating state sequence of length m, there is (effectively) a $(1,1)$ reaction system with $\sharp S = 3(k+1)$ and with a terminating state sequence of length $2m + 3$. Thus, we have a sequence of numbers $u_k,\ k = 1, 2, \ldots$, defined by the recursion

$$u_1 = 5,\ u_{k+1} = 2u_k + 3,\ k \geq 1.$$

Solving this linear recursion, [4], we get the values explicitly:

$$u_k = 4 \cdot 2^k - 3, \ k \geq 1.$$

Hence we obtain the following result, applying also the last sentence of Lemma 4.

Theorem 3. *For all $k \geq 1$, there is a $(1,1)$ reaction system with the cardinality of the state $3k$ and with a terminating state sequence of length $4 \cdot 2^k - 3$, as well as a $(1,1)$ reaction system with the cardinality of the state set $3k+3$ and with a cycle of length $4 \cdot 2^k - 1$.*

Exponentially better estimates for the lengths are obtained by considering the recursion based on the following lemma.

Lemma 5. *Assume that the base set S of a $(1,1)$ reaction system $\mathcal{A}$ is of cardinality n and that $\mathcal{A}$ has a terminating state sequence of length $m \geq 1$. Then there is another $(1,1)$ reaction system $\mathcal{A}_1$, with the base set of cardinality $n+4$ and with a terminating state sequence of length $3m+5$.*

Proof. Assume that the terminating state sequence of $\mathcal{A}$ is

$$Z_0 \Rightarrow Z_1 \Rightarrow \ldots \Rightarrow Z_{m-1} \Rightarrow Z_m = \phi.$$

The base set of the reaction system $\mathcal{A}_1$ is $S \cup \{a,b,c,d\}$, where a,b,c,d are new elements. The system $\mathcal{A}_1$ has all the reactions of $\mathcal{A}$ and, in addition, the following reactions:

$$\rho_{1i} = (i,d,b), \text{ for all } \ i \in S, \ \rho_{2i} = (i,b,d), \text{ for all } i \in S,$$
$$\rho_3 = (b,c,a), \ \rho_4 = (a,b,\{Z_0,b,c\}), \ \rho_5 = (b,a,c), \ \rho_6 = (c,b,\{Z_0,d\}).$$

We now claim that the state sequence starting with $Z_0 \cup \{a,b\}$ terminates and is of length $3m+5$.

Indeed, the beginning of the sequence

$$Z_0 \cup \{a,b\} \Rightarrow Z_1 \cup \{a,b\} \Rightarrow \ldots \Rightarrow Z_{m-1} \cup \{a,b\} \Rightarrow \{a,b\} \Rightarrow \{a\} \Rightarrow Z_0 \cup \{b,c\}$$

is of length $m+2$. The reaction ρ_{1i} keeps b alive, and ρ_3 keeps a alive. Neither ρ_{2i} nor ρ_5 is applicable, whereas ρ_4 is applicable only at the last step.

The continuation

$$Z_0 \cup \{b,c\} \Rightarrow Z_1 \cup \{b,c\} \Rightarrow \ldots \Rightarrow Z_{m-1} \cup \{b,c\} \Rightarrow \{b,c\} \Rightarrow \{c\} \Rightarrow Z_0 \cup \{d\}$$

is also of length $m+2$. Again ρ_{1i} keeps b alive but now ρ_5 keeps c alive. The reaction ρ_{2i} is still not applicable. Now ρ_3 is not applicable, and ρ_6 is applicable at the last step only.

The final part of the sequence

$$Z_0 \cup \{d\} \Rightarrow Z_1 \cup \{d\} \Rightarrow \ldots \Rightarrow Z_{m-1} \cup \{d\} \Rightarrow \{d\} \Rightarrow \phi$$

is of length $m+1$, whence the total length of the terminating sequence is $3m+5$. Now ρ_{2i} keeps d alive, whereas ρ_{1i} is not applicable. □

Consider now the $(1,1)$ reaction system with the base set $\{1,2,3,4\}$ and reactions

$$(1,3,2),\ (2,3,3),\ (3,1,4),\ (4,3,14).$$

We obtain the terminating state sequence

$$1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 14 \Rightarrow 124 \Rightarrow 1234 \Rightarrow \phi$$

of length 7.

Thus, for $k = 1$, there is a $(1,1)$ reaction system with $\sharp S = 4k$ and with a terminating state sequence of length 7. According to Lemma 5, whenever there is a $(1,1)$ reaction system with $\sharp S = 4k$ and with a terminating state sequence of length m, there is (effectively) a $(1,1)$ reaction system with $\sharp S = 4(k+1)$ and with a terminating state sequence of length $3m+5$. Thus, we now have a sequence of numbers $v_k,\ k = 1,2,\ldots$, defined by the recursion

$$v_1 = 7,\ v_{k+1} = 3v_k + 5,\ k \geq 1.$$

This gives us the solution

$$v_1 = 7,\ v_2 = 26,\ v_k = 3 \cdot 3^k + 9(3^{k-3} - 1)/2 + 2,\ k \geq 3.$$

Hence, we have established the following result.

Theorem 4. *For every $k \geq 1$, there is a $(1,1)$ reaction system having $4k$ elements in the base set and having a terminating state sequence of length v_k.*

Applying the last sentence of Lemma 4, we obtain the further result.

Theorem 5. *For every $k \geq 1$, there is a $(1,1)$ reaction system having $4k+3$ elements in the base set and having a cycle of length $v_k + 2$.*

It is unlikely that one is able to get even close to the optimal length $2^{\sharp S}$ of terminating sequences or cycles. However, better methods might yield essentially better estimates than the ones so far obtained.

5 Conclusion

We have considered in this paper certain formal combinatorial properties of reaction systems, focusing on the basic variant of the systems. The amazing diversified applicability of reaction systems has made it necessary to introduce many modifications of the basic variant. However, the basic variant itself is worth of further study, especially because there still remain interesting open problems and problem areas. We have seen that the interactive processes with unlimited auxiliary sequences yield "everything". An open problem is to study cases where the auxiliary sequences are somehow limited, for instance, periodic.

Reaction systems can also be used as generators of formal languages, although no significant results in this direction exist so far.

Theoretically most interesting problems concern $(1, 1)$ reaction systems, in particular, the characterization of their sequences. Denote by $\mu(n)$ the maximal length of a terminating sequence generated by a reaction system with $\sharp S = n$. An interesting combinatorial problem is to investigate the function $\mu(n)$ and find good bounds for it.

References

1. Brijder, R., Ehrenfeucht, A., Main, M., Rozenberg, G.: A tour of reaction systems. International Journal of Foundations of Computer Science (to appear)
2. Ehrenfeucht, A., Main, M., Rozenberg, G.: Functions defined by reaction systems. International Journal of Foundations of Computer Science 22, 167–178 (2011)
3. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. Fundamenta Informaticae 76, 1–18 (2006)
4. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, New York (1980)
5. Salomaa, A.: Mirror images and schemes for the maximal complexity of nondeterminism. Fundamenta Informaticae (to appear)
6. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoretical Computer Science 320, 293–313 (2004)

On Distance Coloring

A Review Based on Work with Dexter Kozen

Alexa Sharp

Oberlin College, Oberlin, OH

Abstract. An undirected graph $G = (V, E)$ is (d, k)-*colorable* if there is a vertex coloring using at most k colors such that no two vertices within distance d have the same color. It is well known that $(1, 2)$-colorability is decidable in linear time, and that $(1, k)$-colorability is *NP*-complete for $k \geq 3$. This paper presents the complexity of (d, k)-coloring for general d and k, and enumerates some interesting properties of (d, k)-colorable graphs. The main result is the dichotomy between polynomial and *NP*-hard instances: for fixed $d \geq 2$, the distance coloring problem is polynomial time for $k \leq \lfloor \frac{3d}{2} \rfloor$ and NP-hard for $k > \lfloor \frac{3d}{2} \rfloor$. We present a reduction in the latter case, as well as an algorithm in the former. The algorithm entails several innovations that may be of independent interest: a generalization of tree decompositions to overlay graphs other than trees; a general construction that obtains such decompositions from certain classes of edge partitions; and the use of homology to analyze the cycle structure of colorable graphs. This paper is both a combining and reworking of the papers of Sharp and Kozen [14,10].

1 Introduction

Given a connected undirected graph $G = (V, E)$, the classic k-*coloring* problem assigns a color from 1 to k to each vertex in a graph such that no two adjacent vertices share the same color [6]. The k-coloring problem, along with many variations and generalizations, is well-studied in both computer science and mathematics [2,4,1,15,3]. Its applications range from frequency assignment [5] to circuit board testing [7], among others.

The *distance* (d, k)-*coloring* problem is a generalization of k-coloring that again assigns a color from 1 to k to each vertex, but such that no two vertices within distance d of each other share the same color. Clearly, k-coloring is a special case of (d, k)-coloring with $d = 1$. It is well known that $(1, k)$-coloring is *NP*-complete for $k \geq 3$ [6] and solvable in linear time for $k \leq 2$: a graph is $(1, 2)$-colorable if and only if it has no odd cycles and $(1, 1)$-colorable if and only if it has no edges. Furthermore, (d, k)-coloring a graph G is equivalent to k-coloring G^d, the dth power graph of G. (The graph G^d has the same vertex set as G and an edge between two vertices if and only if they are within distance d of each other in G.) In this way (d, k)-coloring is no harder than $(1, k)$-coloring.

These results still leave room for questions, however; specifically, for which values of d and k can (d, k)-coloring be solved in polynomial-time? The dichotomy

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 283–297, 2012.

between polynomial and NP-hard instances is the subject of this paper: the main result is a sharp boundary that classifies the complexity of all instances of (d,k)-coloring for $d \geq 2$: determining whether a graph is (d,k)-colorable is polynomial-time for $k \leq \lfloor \frac{3d}{2} \rfloor$, but NP-hard for $k > \lfloor \frac{3d}{2} \rfloor$.

The algorithm for $k \leq \lfloor 3d/2 \rfloor$ uses known algorithms for coloring graphs of bounded treewidth [13]. Since the complexity depends so drastically on the treewidth, we want to obtain a bound on treewidth that is as small as possible.

To this end, we describe a construction that achieves treewidth $\leq 2d$. However, the main interest is not the bound itself, but rather the analysis, which reveals a clear picture of the structure of colorable graphs. The construction entails several technical contributions that may be of independent interest:

(i) A generalization of tree decompositions [13] to allow arbitrary overlay graphs, not just trees. In particular, we show that (d,k)-colorable graphs for $k \leq \lfloor 3d/2 \rfloor$ have a *cycle decomposition* of width d.
(ii) A general construction to obtain such decompositions from certain classes of edge partitions. An example of one such partition is the set of biconnected components of an undirected graph, which gives a tree decomposition.
(iii) The use of homology to analyze the cycle structure of colorable graphs. We show that for $k \leq \lfloor 3d/2 \rfloor$, (d,k)-colorable graphs contain at most one "long" cycle modulo "short" cycles.

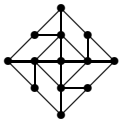


Fig. 1.

On the other side, the reduction and accompanying analysis shed light on a particular source of *NP*-hardness in graph problems, namely, the ability to propagate information *nonlinearly* in the graph. This is a somewhat elusive notion, but intuitively we mean that it must be possible to duplicate information so that it can be transferred to an unbounded number of other locations in the graph. Recall that typical *NP*-hardness proofs in graphs involve the construction of certain "gadgets" that can propagate information in the graph in a controlled way. These gadgets can be quite intricate. For example, to show that planar 3-colorability is *NP*-complete [6], one can use the *crossover gadget* shown in Fig. 1. This gadget has the properties (i) it is planar, (ii) any legal 3-coloring must color the opposite corners the same, and (iii) every 3-coloring of the corners with opposite corners the same extends to a legal 3-coloring of the whole gadget. Thus, by replacing edge crossings with the gadget, 3-colorability of arbitrary graphs can be reduced to 3-colorability of planar graphs. We can think of the gadget in Fig. 1 as propagating coloring information between the east and west corners and simultaneously between the north and south corners.

However, to establish *NP*-hardness, it is not enough just to be able to propagate information in the graph; it must also be possible to *duplicate* information so that it can be propagated *nonlinearly*; that is, it must be possible to transfer the same information to an *unbounded* number of other locations in the graph. Here we are using the term *linear* in the same sense as linear logic [8]. In linear logic, assertions may not be freely duplicated, and every assertion that is

produced must be consumed. Thus the proof rules of linear logic do not allow cost-free nonlinear propagation of information.

This phenomenon arises often in combinatorial problems. For example, two-dimensional matching is tractable, but three-dimensional matching is *NP*-complete [6]. The standard gadget used to prove *NP*-hardness of the latter from 3CNF clearly illustrates the ability to create and propagate information unboundedly.

For another example, in the Boolean satisfiability problem, information is propagated by the constraint that all occurrences of a variable must have the same truth value. The general 3CNF satisfiability problem is *NP*-complete, but the problem is efficiently solvable under the restriction that each variable occur at most twice, or that each clause contain at most 2 variables. Intuitively, these restrictions prevent the nonlinear propagation of information. Allowing three or more occurrences is tantamount to an unbounded number of occurrences by introducing new variables.

Likewise, in the distance-d coloring problem, allowing more than $\lfloor 3d/2 \rfloor$ colors essentially enables the nonlinear propagation of information. A paradigmatic example is the graph illustrated in Fig. 2 with $d = 6$. This graph is $(6, 10)$-colorable, and any legal coloring imposes certain constraints on the colors of any node connected to one of the three terminal nodes. In the $(6, 10)$-coloring problem, and in fact for all (d, k)-coloring problems for $d \geq 2$ and $k > \lfloor 3d/2 \rfloor$, we can build gadgets based on this idea, and these gadgets can be composed to propagate coloring constraints nonlinearly, and these problems are *NP*-complete. (See Section 2.)

Fig. 2.

On the other hand, the graph of Fig. 2 is not $(6, 9)$-colorable, so it cannot appear as a subgraph of any $(6, 9)$-colorable graph. This severely restricts the form of $(6, 9)$-colorable graphs, and similarly of (d, k)-colorable graphs for $k \leq \lfloor 3d/2 \rfloor$, to the extent that they are efficiently recognizable and efficiently colorable.

This paper is organized as follows. Section 2 describes the reduction of [14]. The rest of the paper is based on [10]: Section 3 describes a generalization of tree decompositions and explains how to obtain such decompositions from a certain class of edge partitions, of which biconnected components are a special case. Section 4 shows that for $k \leq \lfloor 3d/2 \rfloor$, (d, k)-colorable graphs have a tree decomposition of width at most $2d$ that we find efficiently. The homology of colorable graphs is treated in Section 5.

2 NP-Hardness

We know that $(1, k)$-coloring is NP-hard for $k \geq 3$ [6]; this section shows that for $d \geq 2$, (d, k)-coloring is NP-hard for $k > \lfloor \frac{3d}{2} \rfloor$ [14].

Theorem 1. *The (d, k)-coloring problem is NP-hard for $d \geq 2$, $k > \lfloor \frac{3d}{2} \rfloor$.*

Theorem 1 follows via a reduction from k-coloring. Given an instance (G, k) of k-coloring, we construct a graph G' such that G is k-colorable if and only if G' is (d, k)-colorable. The building block of this reduction is a *triangle gadget*

G^Δ, shown in Figure 3 for odd and even d. The gadget is such that G^Δ is (d,k)-colorable if and only if x, y and z have the same color.

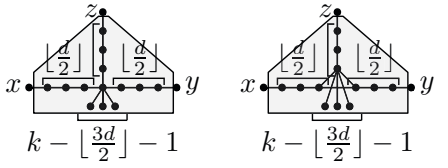


Fig. 3.

We define $G' = (V', E')$ as follows. For each vertex $u \in V$, create gadget $G^u \subseteq G'$ by concatenating $deg(u) \cdot 2k^4$ copies of G^Δ, overlapping the x and y vertices, leaving the z vertices open. Every $2k^4 th$ z vertex is reserved for use as follows: for each edge $e = (u, v) \in G$, create an edge $(u_e, v_e) \in G'$ where u_e and v_e are reserved z vertices of G^u and G^v, respectively; an example of this reduction is shown in Fig. 4. Note that G' is polysize, as $|G^\Delta| = k + 2$ and it is copied $\sum_{u \in V} deg(u) \cdot 2k^4$ times.

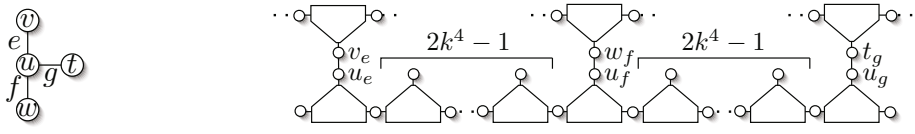


Fig. 4.

Note that if G' is (d,k)-colorable, then $u_e, u_f \in G'$ have the same color for all edges e, f incident to $u \in G$, and for edge $e = (u, v) \in G$, $u_e, v_e \in G'$ are different colors. Together these imply that G is k-colorable.

To finish the reduction, it remains to argue that every k-coloring in G corresponds to a (d,k)-coloring in G'. This direction is less straightforward; the reader is referred to [14] for the details.

3 Decompositions and Forbidden Subgraphs

This section introduces a generalization of tree decompositions [13] such that the "overlay graph" is not restricted to a tree.

Let i, j, k be vertices of an undirected graph $G = (V, E)$. The set $[i, k]$ is the set of all vertices lying on all shortest paths from i to k, inclusive, in G. The set $[i, k]$ is called an *interval*. We say that j is *between* i *and* k if $j \in [i, k]$.

Definition 1. *A* decomposition *of* G *is a triple* (W, F, X) *consisting of an undirected* overlay graph (W, F) *and a map* $X : W \to 2^V$ *associating a subset* $X_i \subseteq V$ *with each* $i \in W$ *satisfying the following properties.*

(i) *The* X_i *cover* V*; that is,* $V = \bigcup_{i \in W} X_i$.
(ii) *For all edges* $(u, v) \in E$*, there exists* $i \in W$ *such that* $u, v \in X_i$.
(iii) *If* j *is between* i *and* k *in* (W, F)*, then* $X_i \cap X_k \subseteq X_j$.

The width *of* (W, F, X) *is* $\max_i |X_i| - 1$.

A *tree decomposition* is just a decomposition in which the overlay graph (W, F) is an undirected tree. We also consider *cycle decompositions* and *line decompositions* in which the overlay graphs are undirected cycles and paths, respectively.

3.1 Decompositions from Edge Partitions

Section 4 uses the following decomposition, formed by an edge partition whose partition elements share at most one vertex (called an *articulation point*):

Lemma 1. *If $P \subseteq 2^E$ is an edge partition of $G = (V, E)$ such that $|V(A) \cap V(B)| \leq 1$ for all $A, B \in P$, then (W, F, X) is a decomposition of G, where*

$$\begin{aligned} W &= P \cup \{u \mid V(A) \cap V(B) = \{u\} \text{ for some } A, B \in P\} \\ F &= \{(u, A) \mid u \in V(A)\} \\ X_A &= V(A) \\ X_u &= \{u\}. \end{aligned}$$

Proof. Properties (i) and (ii) follow from the fact that G is connected and P is an edge partition. For (iii), X_u and X_v cannot intersect for $u \neq v$. If X_u and X_A intersect, then $(u, A) \in F$, and there is no other vertex of (W, F) between u and A, therefore (iii) holds vacuously. Finally, if X_A and X_B intersect for $A \neq B$, then by our premise, $X_A \cap X_B = \{u\}$ for some vertex u, thus $[A, B] = \{A, u, B\}$, and $X_A \cap X_B = \{u\} = X_u$.

Example 1. One such decomposition is given by the biconnected components of G and their articulation points [9]. The biconnected components are the equivalence classes of edges defined by the equivalence relation: $e \sim e'$ if e and e' lie on a common simple cycle. In this case, the decomposition provided by Lemma 1 is a tree decomposition of G, and its width is the maximum size of a biconnected component minus 1. We generalize this construction in Section 4.2.

3.2 Forbidden Subgraphs

For a subgraph $G' \subseteq G$, the *diameter of G'*, denoted $\operatorname{diam} G'$, is the maximum shortest path distance between any two vertices of G'. A *forbidden subgraph* is a subgraph $G' = (V', E')$ such that $\operatorname{diam} G' \leq \min\{d, |V'| - (k - d) - 1\}$.

Theorem 2. *If $G = (V, E)$ is (d, k)-colorable and $|V| \geq k + 1$, then G cannot contain a forbidden subgraph.*

Proof. Given a forbidden subgraph G', let G'' be a connected graph induced by V' and $\max\{0, k + 1 - |V'|\}$ vertices of $V \setminus V'$; G'' has $\geq k + 1$ vertices and diameter at most d, which precludes a (d, k)-coloring.

Example 2. One such forbidden subgraph for any $k \leq \lfloor \frac{3d}{2} \rfloor$ is a central vertex connected to two paths of length $\lfloor d/2 \rfloor$ and one path of length $\lceil d/2 \rceil$ (as in Fig. 2). This subgraph occurs when there is a cycle of length $\geq d + 1$ with an offshoot of length $\geq \lfloor d/2 \rfloor$.

4 Algorithm

Let $k, d \geq 2$ be fixed constants, $k \leq \lfloor 3d/2 \rfloor$. This section describes an algorithm that either declares G not (d,k)-colorable, or constructs a constant-width tree, line, or cycle decomposition of G, which is then transformed into a tree decomposition. We can then apply known algorithms for coloring graphs of bounded treewidth [11,12] to either produce a coloring of G or declare that no such coloring exists. The algorithm is linear in the size of G for fixed d. The algorithm proceeds in several steps, which we treat in the indicated subsections.

§4.1 We find the biconnected components of G. If all such components have diameter $\leq d$ then we either determine that G is not (d,k)-colorable, or produce a tree decomposition of width $k \leq \lfloor 3d/2 \rfloor$.

§4.2 Otherwise G contains exactly one biconnected component G' of diameter $\geq d+1$, for which we construct a cycle decomposition by defining an edge partition then using Lemma 1.

§4.3 We use this cycle decomposition of G' to find a cycle decomposition of G of width at most $k - \lfloor d/2 \rfloor \leq d$.

§4.4 Finally, we construct a line decomposition of width $2d$ from our cycle decomposition of width d by collapsing the two strands of the cycle.

Any one of these steps may fail if G is not colorable, but this will occur in a recognizable way.

4.1 Biconnected Components

In linear time we find the biconnected components of G [9]. There are 3 cases:

(i) All such components have diameter $\leq 3d/4$. If at least one component contains $> k$ vertices, then G is not (d,k)-colorable. Otherwise, these biconnected components and their articulation points provide a tree decomposition of G of width at most $k \leq \lfloor 3d/2 \rfloor$, as in Example 1.

(ii) There is a biconnected component G' of diameter $\leq d$ (but $> 3d/4$). Then G' (hence G) is not (d,k)-colorable, since there is a simple cycle of length greater than $\lfloor 3d/2 \rfloor \geq k$ in a set of diameter $\leq d$.

(iii) There is a biconnected component G' of diameter $\geq d+1$. Then there can be only one (otherwise we could construct a forbidden subgraph out of these components and their connection points, as in Example 2). The following section produces a cycle decomposition in this case.

4.2 A Cycle Decomposition

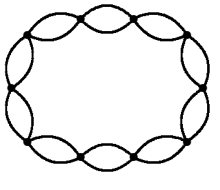


Fig. 5.

Suppose we have one biconnected component G' of diameter at least $d+1$. This implies the existence of a "shortest" simple cycle T (the *trunk*) in G' of length at least $2d+2$. We show how to find a cycle decomposition of G' of small width, provided G' is colorable. The picture to keep in mind is Fig. 5, which represents G', its smaller biconnected components and their "local" articulation points.

First we construct a suitable edge partition consisting of the equivalence classes of a certain equivalence relation $\overset{d}{\sim}$. This partition is very much like that of Example 1, except that we impose a bound of d on the length of cycles. The following technical lemma is needed for transitivity.

Lemma 2. *If G' is (d,k)-colorable, then G' has no simple cycle of length ℓ for any $d+1 \le \ell \le 2d+1$.*

Proof. By way of contradiction, suppose G' contains a simple cycle L of length ℓ, $d+1 \le \ell \le 2d+1$. Then L contains ℓ vertices within distance $\operatorname{diam} L = \lfloor \ell/2 \rfloor \le d$. Note, therefore, that $\ell \le k \le \lfloor 3d/2 \rfloor$.

Note that if T and L are disjoint, then since G' is connected, T and L would be connected by an isthmus, and we could construct a forbidden subgraph as in Example 2. Thus T and L have a node in common.

Let $Q_1, Q_2, \ldots, Q_m$ be the maximal (contiguous) segments of $T \setminus L$. There are $2c$ nodes on Q_i of distance c from L, for $c \le \lfloor |Q_i|/2 \rfloor$. Consider the set $P \subseteq T \setminus L$ of the $\lfloor d/2 \rfloor$ closest vertices to L. Then if $q_i = |P \cap Q_i|$, then $q_1 + q_2 + \cdots + q_m = \lfloor d/2 \rfloor$; moreover, the distance from any $x_i \in P \cap Q_i$ to any $x_j \in P \cap Q_j$ is at most $d(x_i, L) + \lfloor \ell/2 \rfloor + d(x_j, L) \le q_i/2 + \lfloor 3d/4 \rfloor + q_j/2 \le d/4 + 3d/4 \le d$. Thus there are $\ell + \lfloor d/2 \rfloor \ge k+1$ vertices within distance d, a contradiction.

For edges e, e' in G', define $e \overset{d}{\sim} e'$ if e and e' lie on a simple cycle of length at most d. This definition is identical to the definition of the edge partition for biconnected components except for the length restriction.

Lemma 3. *If G' is colorable, then the relation $\overset{d}{\sim}$ is an equivalence relation and no two $\overset{d}{\sim}$-equivalence classes have more than one edge endpoint in common.*

Proof. Reflexivity follows from the fact that any edge and its two endpoints constitute a simple cycle. The relation is symmetric, since e and e' can be interchanged in the definition of $\overset{d}{\sim}$. For transitivity, suppose $(u,v) \overset{d}{\sim} (u',v')$ and $(u',v') \overset{d}{\sim} (u'',v'')$, due to cycles C and C', respectively, both of length $\le d$. Then we can certainly construct a simple cycle D of length $\le 2d$ from C and C' that contains (u,v) and (u'',v''); by Lemma 2, this cycle in fact has length $\le d$.

Lastly, suppose that C and C' are two equivalence classes, and $u \ne v$ are elements of $V(C) \cap V(C')$. Then u and v are at most distance $d/2$ in C and in C'. Combining shortest paths in C and C' between u and v yields a simple cycle of length at most d with edges from both C and C', thus $C = C'$.

Lemma 3 allows us to apply the theory of Section 3. Thus the $\overset{d}{\sim}$-equivalence classes give a decomposition of G' of width at most k. Later, in Section 4.3, we will improve the width bound. First we show that the overlay graph is a cycle.

To do this, we need to identify local articulation points on T, that is, vertices v such that $G' \setminus v$ is not biconnected. A *chord with endpoints* $s, t \in T$ is a path between s and t, none of whose intermediate nodes lie on T. A node $u \in T$ is *subtended* by a chord if it lies strictly between the chord's endpoints on the shortest path between them. No local articulation point is subtended by a chord.

Lemma 4. *If G' is (d,k)-colorable then there are at least 4 local articulation points on T.*

Proof. If a node x on T is subtended by a chord, then the two edges on T adjacent to x are $\overset{d}{\sim}$-equivalent by Lemma 2, because the chord forms a cycle with some portion of T. If every node of T were subtended by a chord, then by transitivity, all edges of T would be $\overset{d}{\sim}$-equivalent, a contradiction to $\operatorname{diam} G' \geq d+1$. Thus there exists some node x not subtended by any chord, the first articulation point.

Similarly, there is an articulation point y within distance $\lfloor d/2 \rfloor$ of x on T, otherwise the shortest path between them in the biconnected component would be shorter than the path on T (a contradiction to T being "shortest"). Since $|T| \geq 2d+2$, there are at least four articulation points total.

Lemma 5. *If G' is (d,k)-colorable, then G' consists of a cycle of biconnected subgraphs, each of diameter at most $\lfloor d/2 \rfloor$. Each biconnected subgraph is connected to its two neighbors on the cycle by a local articulation point. We can find the decomposition in linear time.*

Proof. The existence is clear from Lemma 4. To find the decomposition, we identify the local articulation points. This can be done in linear time: performing a breadth-first search from a vertex s down to a depth of $\lfloor d/2 \rfloor$ can confirm whether s is a local articulation point by whether it is subtended by a chord; since every set of size $\lfloor d/2 \rfloor$ must contain a point of T, and there is a local articulation point no more than $\lfloor d/2 \rfloor$ from any node of T, the first articulation point s is easily found. We break the graph at s, forming two copies of s, then find the biconnected components of the resulting graph.

4.3 Bounding Cycle Width

Section 4.2 constructs an edge partition P of G' that forms a cycle decomposition; this section augments P to a cycle decomposition of G of width at most $k - d/2 \geq 0$ (if $k < d/2$ then no connected subgraph of $d/2$ vertices is colorable.)

Consider any element $A \in P$ (each a biconnected subgraph of diameter at most $d/2$) and its 2 articulation points on T that surround it. Let T_A be the interval of T (of length $\leq d/2$) that contains the 2 articulation points of A and let N_A be all the nodes of G connected to T_A through only nodes in $G \setminus T$. In this way, T_A and N_A are connected, no edges go between N_A and $T \setminus T_A$ (because T_A contains the articulation points of A and G' is maximally biconnected), and all nodes and edges of G are contained within some subgraph $N_A \cup T_A$ for some $A \in P$. The goal of this section is to show that the $\{N_A \cup T_A : A \in P\}$ and the corresponding articulation points form a cycle decomposition of G of width at most $k - d/2$. Properties (i), (ii) and (iii) hold given the previous observations. The width follows from the following lemma.

Lemma 6. *If G is (d,k)-colorable, then $|N_A \cup T_A| \leq k - d/2$.*

Proof. Notice that $|N_A \cup T_A| = |N_A| + |T_A| \leq |N_A| + d/2 + 1$, which means that we only need to show that $|N_A| \leq k - d - 1$ in order to finish the proof.

The idea is to add a carefully chosen $d + 1$ vertices to N_A such that the diameter of the resulting subgraph is still $\leq d$; thus, the subgraph contains $\leq k$ vertices, and N_A at most $k - d - 1$ vertices. To choose these $d + 1$ vertices, we identify a "central" vertex $z \in T_A$ such that all $u \in N_A$ are of distance at most $\lfloor d/2 \rfloor$ from z; thus the vertices of N_A along with the path of length d centered on z will be a subgraph of diameter d containing $d + 1 + |N_A|$ vertices.

Thus it remains to find a $z \in T_A$ such that $d(z, u) \leq \lfloor d/2 \rfloor$ for each $u \in N_A$. To this end, let P be a simple "U-shaped" path in $N_A \cup T_A$ from some $x \in N_A$ to a $t_x \in T_A$ to some $t_y \in T_A$ to some $y \in N_A$. Without loss of generality, let t_x be to the left of t_y on T_A. Then if the $d_P(x, y) = d$, the subgraph containing P, the $d_P(x, t_x)$ vertices of T to the left of t_x and $d_P(y, t_y)$ vertices to the right of t_y form a subgraph of diameter d containing $2|P| - d_{T_A}(t_x, t_y) + 1$ vertices. Since $d_{T_A}(t_x, t_y) \leq \lfloor d/2 \rfloor$, this subgraph contains at least $\lceil 3d/2 \rceil + 1 \geq k + 1$ vertices within diameter d, which is not possible in a (d, k)-colorable graph. Thus, if G is (d, k)-colorable then all such "U-shaped" paths have length at most $d - 1$.

For each vertex $x \in N_A$, let h_x denote the length of the longest simple path from x to some $t_x \in T_A$, and let d_x denote the distance from t_x to the midpoint of T_A. Note that $h_x \leq \lfloor d/2 \rfloor$ for all $x \in N_A$ otherwise we would have a forbidden subgraph. Let $w = argmax_{x \in N_A}\{h_x - (\lfloor d/2 \rfloor - d_x)\}$. If all $h_x - (\lfloor d/2 \rfloor - d_x) < 0$ then the midpoint of T_A works as our z. For $h_w - (\lfloor d/2 \rfloor - d_w) \geq 0$, without loss of generality, suppose that t_w is on the left half of T_A. Let $z \in T_A$ be the vertex that is $h_w - (\lfloor d/2 \rfloor - d_w)$ to the left of the midpoint of T_A. Then note that the longest simple path from w to z is $d_w - d_z + h_w = \lfloor d/2 \rfloor$. Now any vertex u to the right of z defines a U-shaped path between w and u of length $h_w + d_w - d_z + d_u + d_z + h_u = \lfloor d/2 \rfloor + d_u + d_z + h_u$; since this U-shaped path has length at most $d-1$, we have that the distance from u to z is at most $\lfloor d/2 \rfloor$. Any vertex u to the left of z is such that $h_u + d_u \leq h_w + d_w$ by our choice of w; thus $d_u + h_u - d_z \leq d_w + h_w - d_z = \lfloor d/2 \rfloor$. Therefore, every vertex $u \in N_A$ is of distance at most $\lfloor d/2 \rfloor$ from z, and therefore $N_A \cup T_A$ has diameter at most d. If we add $d + 1$ nodes on T centered on z, we have a subgraph of diameter d with $|N_A| + d + 1$ nodes, as desired.

4.4 Converting Cycles to Lines

Section 4.3 shows that G has a cycle decomposition of width at most $k - d/2$. The main algorithmic result is now immediate from the following lemma.

Lemma 7. *If G has a cycle decomposition of width w, then G has a line decomposition of width at most $2w$.*

Proof. Intuitively, we grasp the cycle by diametrically opposed nodes and pull, creating a line with two strands.

Formally, let (C, F, X) be a cycle decomposition of $G = (V, E)$ of width w such that (C, F) on the vertices $0, 1, 2, \ldots, m$, in this order. Consider

$$\begin{aligned} L &= \{0, 1, 2, \ldots, \lfloor m/2 \rfloor\}, \\ D &= \{(i, i+1) \mid 0 \leq i \leq \lfloor m/2 \rfloor - 1\}, \\ Y_i &= X_i \cup X_{m-i}. \end{aligned}$$

We wish to show that (L, D, Y) is a line decomposition of G of width at most $2w$. Properties (i) and (ii) of line decompositions hold trivially, and it is clearly of width at most $2w$. It remains to show property (iii); that is, for $0 \leq i < j < k \leq \lfloor m/2 \rfloor$ that $Y_i \cap Y_k \subseteq Y_j$, or

$$(X_i \cup X_{m-i}) \cap (X_k \cup X_{m-k}) \subseteq X_j \cup X_{m-j} \ .$$

By distributivity this reduces to the following four inclusions:

$$X_i \cap X_k \subseteq X_j \cup X_{m-j} \tag{1}$$
$$X_i \cap X_{m-k} \subseteq X_j \cup X_{m-j} \tag{2}$$
$$X_{m-i} \cap X_k \subseteq X_j \cup X_{m-j} \tag{3}$$
$$X_{m-i} \cap X_{m-k} \subseteq X_j \cup X_{m-j}. \tag{4}$$

These inclusions hold by virtue of the fact that (C, F, X) is a cycle decomposition. Properties (1) and (4) are immediate, since $j \in [i, k]$ and $m - j \in [m - i, m - k]$. For (2) and (3), there are two cases, depending on whether

$$\begin{aligned} &(m - k) - i \bmod m + 1 \leq \lfloor (m + 1)/2 \rfloor \text{ or} \\ &i - (m - k) \bmod m + 1 \leq \lfloor (m + 1)/2 \rfloor. \end{aligned}$$

In the former case, $k \in [i, m - k]$, therefore $j \in [i, m - k]$ and $m - j \in [m - i, k]$. These imply that $X_i \cap X_{m-k} \subseteq X_j$ and $X_{m-i} \cap X_k \subseteq X_{m-j}$.

In the latter case, we have $k - (m - i) \bmod m + 1 \leq \lfloor (m + 1)/2 \rfloor$. Then $i \in [m - i, k]$, therefore $j \in [m - i, k]$ and $m - j \in [i, m - k]$. These imply that $X_{m-i} \cap X_k \subseteq X_j$ and $X_i \cap X_{m-k} \subseteq X_{m-j}$.

Theorem 3. *If G is (d, k) colorable for $k \leq \lfloor 3d/2 \rfloor$, then G has a line decomposition of width at most $2k - d \leq 2d$.*

5 Homology of Colorable Graphs

Colorability has deeper implications regarding the cycle structure of a graph that are best expressed in terms of homology. The main observation is Theorem 4, which is a formal statement of the intuitive idea that all long simple cycles in the graph are equivalent modulo short cycles; in other words, modulo short cycles, there is at most one long simple cycle in the graph. The formal statement requires some number of definitions.

A free abelian group H on a finite set of generators X is the set of formal sums $\sum_{e\in X} k_e e$, where $k_e \in \mathbb{Z}$ are integer coefficients. This also forms a $\mathbb{Z}$-module of dimension $|X|$ with standard basis X. We have a natural inner product defined by $e \bullet e' = 1$ if $e = e'$ and 0 if $e \neq e'$ for basis elements $e, e' \in X$, extended uniquely to $H^2 \to \mathbb{Z}$ by bilinearity. For a basis element $e \in X$, the projection $x \mapsto e \bullet x$ gives the coefficient of e in the expression x.

For a connected undirected graph $G = (V, E)$, let $\boldsymbol{E}$ be the set of *directed* edges of G, consisting of two directed edges uv and vu for each undirected edge $\{u, v\} \in E$. Let $\boldsymbol{\Delta}$ be the set of oriented simple cycles, where $\langle u_1 u_2 \cdots u_n \rangle$ is the oriented cycle through $u_1, u_2, \ldots, u_n$ in that order. If e is a directed or undirected edge with endpoints u, v, let $\langle e \rangle = \langle uv \rangle \in \boldsymbol{\Delta}$ be the cycle of length 2 consisting of the two directed edges uv and vu. An element of $\boldsymbol{\Delta}$ is *short* if its length is at most $2d + 1$, otherwise it is *long*. Let $\boldsymbol{\Delta}^d$ be the set of short oriented simple cycles. Let H_0, H_1, and H_2 be the free abelian groups on generators V, $\boldsymbol{E}$, and $\boldsymbol{\Delta}$, respectively. Let H_2^d be the subgroup of H_2 generated by $\boldsymbol{\Delta}^d$.

For an object α in $\boldsymbol{\Delta}$ or $\boldsymbol{E}$, let $\overline{\alpha}$ denote the object with the opposite orientation. For example, if α is the oriented simple cycle $\langle uvw \rangle$, then $\overline{\alpha} = \langle wvu \rangle$, and if e is the directed edge uv, then $\overline{e} = vu$. The maps $\overline{\ } : \boldsymbol{\Delta} \to \boldsymbol{\Delta}$ and $\overline{\ } : \boldsymbol{E} \to \boldsymbol{E}$ extend by linearity to group homomorphisms $\overline{\ } : H_2 \to H_2$ and $\overline{\ } : H_1 \to H_1$, respectively.

Let $\partial_1 : H_2 \to H_1$ and $\partial_0 : H_1 \to H_0$ be the *boundary homomorphisms* defined as follows. For an oriented simple cycle α, $\partial_1(\alpha)$ is the sum of the directed edges in α, and for a directed edge uv, $\partial_0(uv) = v - u$. For example, if α is the oriented simple cycle $\langle uvw \rangle$, then $\partial_1(\alpha) = uv + vw + wu$. For $e \in \boldsymbol{E}$, $\partial_1(\langle e \rangle) = e + \overline{e}$ and $\langle e \rangle = \langle \overline{e} \rangle$.

The *cycle group* $C \subseteq H_1$ is the kernel of ∂_0, which is the same as the image of H_2 under ∂_1 (Lemma 8(i)). Let C^d be the image of H_2^d under ∂_1. The group C^d is the subgroup of the cycle group generated by (the directed edges of) the short oriented simple cycles.

Let P_i be the positive orthant of H_i, $0 \leq i \leq 2$; this is the set of expressions with only nonnegative coefficients. The set P_i induces a partial order $\leq$ on H_i defined by: $\alpha \leq \beta$ iff $\beta - \alpha \in P_i$. An element is *positive* if it is contained in P_i.

Lemma 8. *For P_1, P_2, H_2, C and ∂_1 defined above,*

(i) $\partial_1(H_2) = C$.
(ii) $\partial_1(P_2) = P_1 \cap C$.

Proof. For the forward inclusion of (i), a standard argument shows the image $\partial_1(H_2)$ is contained in $C = \ker \partial_0$; that is, $\partial_0 \circ \partial_1 = 0$. For example, if $\alpha = \langle uvw \rangle$, then $\partial_0(\partial_1(\alpha)) = \partial_0(uv + vw + wu) = (v - u) + (w - v) + (u - w) = 0$.

The forward inclusion of (ii) follows from the observation that the image of a positive element under ∂_1 is positive in H_1.

For the reverse inclusion, we prove (ii) first. This argument is similar to the proof of Euler's theorem that a cycle in an undirected graph can be written as a union of simple cycles. For $x \in P_1$ and $v \in V$, define the *indegree* and *outdegree* of v in x to be the sum of the coefficients in x of all edges entering

v and exiting v, respectively. Observe that for positive x, membership in $\ker \partial_0$ is equivalent to the condition that the indegree of every vertex is equal to its outdegree; equivalently, $v \bullet \partial_0(x) = 0$ for all v.

Let $x \in P_1 \cap \ker \partial_0$. Start at any node of nonzero outdegree, and trace a cycle α_1, following edges with positive coefficients in x until you encounter a node twice. It is always possible to continue, since the outdegree of any node equals the indegree. Subtract off the edges of α_1 between the repeat; the resulting expression $x - \partial_1(\alpha_1)$ is still in $P_1 \cap \ker \partial_0$. Continue in this way to get $\alpha_2, \alpha_3, \ldots$, until the resulting expression is 0. Let $\alpha = \sum_i \alpha_i$. Then $x = \partial_1(\alpha)$ and $\alpha \in P_2$.

For (i), let $x \in \ker \partial_0$. Let $\gamma \in P_2$ be a positive sum of length-2 cycles $\langle e \rangle$ with coefficients just large enough that $x + \partial_1(\gamma) \in P_1$. Then $x + \partial_1(\gamma) \in P_1 \cap \ker \partial_0$. By (ii), there exists $\alpha \in P_2$ such that $\partial_1(\alpha) = x + \partial_1(\gamma)$. Then $\partial_1(\alpha - \gamma) = x$. (For example, the element $uv + vw - uw \in \ker \partial_0$ is $\partial_1(\langle uvw \rangle - \langle uw \rangle)$.)

Theorem 4. *Suppose G is $(d, \lfloor 3d/2 \rfloor)$-colorable for some $d \geq 2$. Then either*

(i) *all simple cycles of G are short, in which case $C^d = C$ and the quotient group C/C^d is trivial; or*
(ii) *G contains a long simple cycle, in which case the quotient group C/C^d is isomorphic to $\mathbb{Z}$.*

Proof. Case (i) is immediate from Lemma 8(i). If all simple cycles are short, then $H_2 = H_2^d$, in which case $C = C^d$.

Case (ii) assumes that G contains an oriented simple cycle T of length at least $2d + 2$. By Lemma 2, G has no simple cycle of length $d + 1$ to $2d + 1$, inclusive; thus C^d is generated by (edges of) simple cycles of length at most d.

We first show that $\partial_1(T)$ is not in C^d, thus $C^d \neq C$ and the quotient C/C^d is nontrivial. By general considerations, we know that it is a $\mathbb{Z}$-module of some finite dimension; we show later that the dimension is 1.

An element of H_2 is *reduced* if it is an expression of the form $\beta + \gamma$, where

(i) $\beta \in P_2$;
(ii) β is orthogonal to H_2^2; that is, $\langle e \rangle \bullet \beta = 0$ for all e;
(iii) $\gamma \in H_2^2$; and
(iv) it is not the case that $\partial_1(\langle e \rangle) \leq \partial_1(\beta)$ for any edge e.

Intuitively, conditions (i)–(iv) say, respectively, that β is positive, β is a sum of cycles of length 3 or greater, γ is a weighted sum of cycles of length 2, and no two cycles of β contain a complementary pair of edges $e, \overline{e}$.

We now claim that for every element of $\alpha \in H_2$, there is a reduced expression $\beta + \gamma \in H_2$ such that $\partial_1(\alpha) = \partial_1(\beta + \gamma)$. Moreover, if $\alpha \in H_2^d$, then we can ensure that $\beta \in H_2^d$; thus the reduction process does not introduce any long cycles.

First, express α as a sum $\beta_1 + \gamma_1$, where β_1 is a weighted sum of cycles of length at least 3 and γ_1 is a weighted sum of cycles of length 2. Then β and γ satisfy (ii) and (iii). To obtain (i), we observe that

$$\partial_1(\alpha + \overline{\alpha}) = \partial_1(\sum_e (e \bullet \alpha)\langle e \rangle);$$

intuitively, all the edges in $\alpha + \overline{\alpha}$ occur in complementary pairs. Thus

$$\partial_1(-\alpha) = \partial_1(\overline{\alpha} - \sum_e (e \bullet \alpha)\langle e \rangle).$$

If $\beta_1 = \beta_1^+ - \beta_1^-$ with $\beta_1^+, \beta_1^- \in P_2$, let

$$\beta_2 = \beta_1^+ + \overline{\beta}_1^- \qquad\qquad \gamma_2 = \gamma_1 - \sum_e (e \bullet \beta_1^-)\langle e \rangle.$$

Then $\partial_1(\alpha) = \partial_1(\beta_2 + \gamma_2)$ and $\beta_2 + \gamma_2$ satisfies (i)–(iii).

Note that none of the operations so far have introduced any long cycles. Thus if $\alpha \in H_2^d$ then $\beta_2 \in H_2^d$.

Finally, to get (iv), suppose $\partial_1(\langle e \rangle) \leq \partial_1(\beta_2)$. Since $\langle e \rangle \bullet \beta_2 = 0$, we must have $\theta, \theta' \in \Delta$ and $e \in \boldsymbol{E}$ such that $\theta + \theta' \leq \beta_2$ and $\partial_1(\langle e \rangle) \leq \partial_1(\theta + \theta')$. We have $\partial_1(\theta + \theta' - \langle e \rangle) \in P_1 \cap C$, so by Lemma 8(ii) there exists $\eta \in P_2$ such that $\partial_1(\eta) = \partial_1(\theta + \theta' - \langle e \rangle)$. Write η as $\eta' + \eta''$, where η' is orthogonal to H_2^2 and $\eta'' \in H_2^2$, and let

$$\beta_3 = \beta_2 - \theta - \theta' + \eta' \qquad\qquad \gamma_3 = \gamma_2 + \eta'' + \langle e \rangle.$$

Then $\partial_1(\alpha) = \partial_1(\beta_3 + \gamma_3)$ and $\beta_3 + \gamma_3$ still satisfies (i)–(iii).

Moreover, $\beta_3 \in H_2^d$ if $\beta_2 \in H_2^d$. To see this, observe that the inner product with $\sum_e e$ gives the sum of the coefficients, which for a cycle gives the length of the cycle. If $\eta = \sum_i \eta_i$, where each $\eta_i \in \Delta$, then because $\theta, \theta' \in \Delta^d$, we have

$$\begin{aligned}
\sum_i ((\sum_e e) \bullet \partial_1(\eta_i)) &= (\sum_e e) \bullet \partial_1(\eta) \\
&= (\sum_e e) \bullet \partial_1(\theta + \theta' - \langle e \rangle) \\
&= (\sum_e e) \bullet \partial_1(\theta + \theta') - (\sum_e e) \bullet \partial_1(\langle e \rangle) \\
&\leq 2d - 2,
\end{aligned}$$

therefore each η_i in the sum η is of length at most $2d - 2$. By Lemma 2, it is of length at most d.

The previous step can be repeated only finitely many times before (iv) becomes satisfied, because each time the inner product of the current β_i with $\sum_e e$ decreases by two. We have shown that every α is equivalent modulo ∂_1 to a reduced $\beta + \gamma$. Moreover, if $\alpha \in H_2^d$, then $\beta \in H_2^d$.

Now we argue that if $\alpha \in \Delta$ and $\partial_1(\alpha) \in C^d$, then $\alpha \in \Delta^d$. This implies that $\partial_1(T) \notin C^d$, since $T \notin \Delta^d$.

Suppose $\alpha \in \Delta$ and $\partial_1(\alpha) \in C^d$. If α is of length 2, there is nothing to prove, so assume α is of length at least 3. There is a reduced form expression $\beta + \gamma$ such that $\partial_1(\alpha) = \partial_1(\beta + \gamma)$, $\beta \in H_2^d$, and $\gamma \in H_2^2$. For any edge $e \in \boldsymbol{E}$,

$$(e - \overline{e}) \bullet \partial_1(\alpha - \beta) = (e - \overline{e}) \bullet \partial_1(\gamma) = 0,$$

since γ is a weighted sum of cycles of length 2; therefore

$$e \bullet \partial_1(\alpha) - \overline{e} \bullet \partial_1(\alpha) = e \bullet \partial_1(\beta) - \overline{e} \bullet \partial_1(\beta).$$

Because α is a simple cycle of length at least 3, each of $e \bullet \partial_1(\alpha)$ and $\overline{e} \bullet \partial_1(\alpha)$ is either 1 or 0, and not both are 1. Also, since β satisfies (iv) of reduced expressions, one of $e \bullet \partial_1(\beta)$ or $\overline{e} \bullet \partial_1(\beta)$ must be 0. It follows by an easy case analysis that we must have $e \bullet \partial_1(\alpha) = e \bullet \partial_1(\beta)$ in all cases. As e was arbitrary, $\partial_1(\alpha) = \partial_1(\beta)$. As α is a simple cycle and β is a positive sum of simple cycles, we must have $\alpha = \beta$.

Now we argue that there is only one long simple cycle modulo short simple cycles. Let T and T' be two oriented long simple cycles. Let a and b be points on T of maximum distance from each other. Let p and q be local articulation points on T such that p is of distance at most $d/2$ from a, moving in the forward direction on T, and q is of distance at most $d/2$ from b, also moving in the forward direction; p and q are at least distance $d + 1 - \lfloor d/2 \rfloor = \lceil d/2 \rceil + 1$ apart on T. Since local articulation points are not subtended by chords, removal of p and q disconnects the graph. Moreover, both T and T' traverse p and q, otherwise we could construct a forbidden subgraph like the one in Example 2. Thus T and T' traverse them exactly once, as they are simple cycles. We can assume without loss of generality that T and T' are oriented so as to traverse them in opposite directions.

Consider $\partial_1(T' + T) \in C$; we show that this expression vanishes modulo C^d. Every edge e of T' is contained in a maximal segment of T' all of whose intermediate nodes do not lie on T. This segment is a chord of T consisting of edges of T' such that its endpoints are distinct and lie on both T and T'. The length of the chord and the length of the segment of T subtended by the chord are both at least 1 and at most $d/2$, otherwise we could construct a forbidden subgraph. Thus these two undirected segments together form a short simple cycle, which we orient in the opposite direction from T'; that is, the cycle contains the edge $\overline{e}$ for every edge e on the chord. The segment of the short cycle coinciding with T is oriented in either the same direction as T or the opposite direction. Let R be the set of short simple cycles obtained in this way.

Now let γ be a maximal sum of cycles of length 2 such that $\partial_1(\gamma) \leq \partial_1(T' + T + \sum R)$. We argue that $\partial_1(T' + T + \sum R - \gamma) = 0$. Consider each edge $e \in \boldsymbol{E}$ separately. If neither e nor $\overline{e}$ is an edge of T' or T, then $e \bullet \partial_1(T' + T + \sum R - \gamma) = 0$. If e lies on T' but neither e nor $\overline{e}$ lies on T, then there is exactly one element of R containing $\overline{e}$ and none containing e. Since both e and $\overline{e}$ appear in $\partial_1(T' + T + \sum R)$ with coefficient 1, $\langle e \rangle$ appears in γ, and

$$e \bullet \partial_1(T' + T + \sum R - \gamma) = e \bullet \partial_1(T' + \sum R - \gamma) = 0.$$

It remains to calculate the coefficient of e and $\overline{e}$ in $\partial_1(T' + T + \sum R - \gamma)$ for $e = uv$ lying on T. Suppose p, u, v, q occur on T in that order. Let P be the segment of T between p and u and let Q be the segment of T between v and q. The segment of T' from q to p traverses an odd number of chords subtending e,

including possibly one of e or $\overline{e}$ itself, and the traversals from Q to P and from P to Q must alternate, with one more traversal from Q to P. Each of these chords is responsible for a cycle of R containing either $\overline{e}$ or e, depending on whether the chord goes from Q to P or from P to Q, respectively. So the coefficients of e and $\overline{e}$ in $\partial_1(\sum R)$ are k and $k+1$, respectively, for some $k \geq 0$. This includes the case in which e or $\overline{e}$ is an edge of T', except the value of k is one greater. Adding in the e from T, we have that the coefficients of e and $\overline{e}$ in $\partial_1(T' + T + \sum R)$ are both $k+1$. Since γ was chosen maximally, $\langle e \rangle$ appears in γ with coefficient $k+1$, therefore the coefficients of e and $\overline{e}$ in $\partial_1(T' + T + \sum R - \gamma)$ are both 0.

Since e was arbitrary, we have $\partial_1(T' + T + \sum R - \gamma) = 0$, so $\partial_1(T + T') = \partial_1(\gamma - \sum R) \in C^d$, therefore $\partial_1(T')$ is equivalent to $-\partial_1(T)$ modulo C^d. We have shown that every element of C is equivalent modulo C^d to kT for some $k \in \mathbb{Z}$.

References

1. Bollobás, B., Harris, A.J.: List-colourings of graphs. Graphs and Combinatorics 1(2), 115–127 (1985)
2. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. The MacMillan Press Ltd. (1978)
3. Chetwynd, A.: Total colourings of graphs. In: Nelson, R., Wilson, R.J. (eds.) Graph Colourings. Pitman Research Notes in Mathematics Series, pp. 65–77. Longman Scientific & Technical, Longman house, Burnt Mill, Harlow, Essex, UK (1990)
4. Fiorini, S., Wilson, R.J.: Edge-colourings of graphs. In: Beineke, L.W., Wilson, R.J. (eds.) Selected Topics in Graph Theory, ch. 5, pp. 103–126. Academic Press, Inc., London (1978)
5. Gamst, A.: Some lower bounds for a class of frequency assignment problems. IEEE Trans. Veh. Technol. VT-35, 8–14 (1986)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1990)
7. Garey, M.R., Johnson, D.S., So, H.C.: An application of graph coloring to printed circuit testing. IEEE Transactions on Circuits and Systems CAS-23(10), 591–598 (1976)
8. Girard, J.-Y.: Linear logic. Theor. Comput. Sci. 50(1), 1–102 (1987)
9. Hopcroft, J., Tarjan, R.E.: Algorithm 447: Efficient algorithms for graph manipulation. Commun. ACM 16(6), 372–378 (1973)
10. Kozen, D., Sharp, A.: On distance coloring. Technical Report cul.cis/TR2007-2084, Cornell University (2007)
11. Makowsky, J.A.: Colored tutte polynomials and kauffman brackets for graphs of bounded tree width. In: Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA 2001), pp. 487–495. SIAM (2001)
12. Noble, S.D.: Evaluating the tutte polynomial for graphs of bounded tree-width. Comb. Probab. Comput. 7(3), 307–321 (1998)
13. Robertson, N., Seymour, P.D.: Graph minors. ii. algorithmic aspects of tree-width. J. Algorithms 7(3), 309–322 (1986)
14. Sharp, A.: Distance Coloring. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 510–521. Springer, Heidelberg (2007)
15. Wilson, B.: Line-distinguishing and harmonious colourings. In: Nelson, R., Wilson, R.J. (eds.) Graph Colourings. Pitman Research Notes in Mathematics Series, pp. 115–133. Longman Scientific & Technical, Longman house, Burnt Mill, Harlow, Essex, UK (1990)

Winning, Losing and Drawing in Concurrent Games with Perfect or Imperfect Information

Glynn Winskel

Computer Laboratory, University of Cambridge, UK

Abstract. Nondeterministic concurrent strategies—those strategies compatible with copy-cat behaving as identity w.r.t. composition—have been characterised as certain maps of event structures. This leads to a bicategory of general concurrent games in which the maps are nondeterministic concurrent strategies. This paper explores the consequences of extending concurrent games with (1) winning, losing and, implicitly, neutral configurations, and (2) access levels, to address situations where Player or Opponent have imperfect information as to what has occurred in the game. In both cases winning strategies are shown to form bicategories of games. The bicategories become equivalent to order-enriched categories when restricted to deterministic strategies.

1 Introduction

Dexter Kozen is an inspiring speaker, enjoys teaching, and has a history of involving students, including undergraduates, in research. We share a close association with the Computer Science department at Aarhus, Denmark, going back over several decades—we first met there in 1979. This paper, in Dexter's honour, is based on a recent student project from a lecture course on concurrent games I gave in Aarhus last summer (August-September, 2011) [1].

Its roots lie in John Conway's "Numbers and Games" [2]. There Conway defined his "surreal numbers" as strengths of certain games: he defined a preorder between games G and H if a winning strategy for G gives rise to a winning strategy for H; the surreal numbers appeared as equivalence classes induced by the preorder. Shortly afterwards André Joyal uncovered a category underpinning Conway's work. Conway's games support two important operations on two-party games: a form of parallel composition $G\|H$, which Conway called a *sum* of games; a dualizing operation $G^{\perp}$ which reverses the roles of Player and Opponent in G, which Conway called *negation*. Joyal, following the method used in Conway's proofs, defined a strategy σ *from* a game G *to* a game H, written $\sigma : G \multimap H$, to be a strategy σ in $G^{\perp}\|H$. Joyal showed that strategies compose, with identities given by copy-cat strategies. A strategy in H corresponds to a strategy from the empty game $\emptyset$ to H. Note that

$$\emptyset \multimap G \multimap H \text{ composes to give } \emptyset \multimap H\,,$$

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 298–317, 2012.

so a strategy in G gives rise to a strategy in H when there is a strategy from G to H. Conway's pre-order between games G and H is witnessed through the presence of a winning strategy of from G to H.

This article takes the ideas of Conway and Joyal into the realm of concurrent/distributed processes. It brings the experience of concurrency (event structures, stable families, their techniques and constructions originally used in the semantics of process languages [3]) to bear on the theory of games. It considers a very general definition of 2-party concurrent games in which Player (more accurately thought of as a team of players) competes against Opponent (a team of opponents) in a potentially highly-distributed fashion, without for instance insisting on the alternation of Player and Opponent moves. For most of the article the games will be games of perfect information, in that Player can see all moves of Opponent, and *vice versa*. An example of such a concurrent game would be simultaneous chess, possibly with collaboration between players. However, the dichotomy Player/Opponent can also be read as process/environment, proof/refutation, or ally/enemy, and there are many other examples of concurrent games in Computer Science and Logic, and beyond.

The methodology is essentially that of Joyal, following Conway, developed within a general model for concurrent computation. Two-party games and strategies are represented as event structures with polarity, in which polarities distinguish the moves of Player and Opponent—*cf.* [4]. A pre-strategy is a total map $\sigma : S \to A$ of event structures with polarity. The map expresses how moves of Player and Opponent, the events of S, correspond to the moves permitted by the game, the events of A; that σ is a map ensures that play of respects the constraints of the game. Following Joyal, a pre-strategy from a game A to a game B is understood as a pre-strategy in a composite game got by setting the dual game of A, reversing the roles of Player and Opponent, in parallel with B. From this general scheme *nondeterministic concurrent strategies*—pre-strategies for which copy-cat strategies behave as identities w.r.t. composition of pre-strategies—have recently been characterized as those pre-strategies which satisfy the two conditions of receptivity and innocence [5]. The extension with winning conditions and the question of when and whether concurrent games are determined (*i.e.* there is either a winning strategy for Player or Opponent) is considered in the Aarhus lecture notes [1] and the forthcoming article [6]. The two contributions of this paper are: (1) an extension of the framework to games with *neutral positions*, which as outcomes of a play yield a draw (the student project at Aarhus); (2) an extension to concurrent games with imperfect information, where moves may be hidden, so cannot be taken account of by strategies. Where the article builds on earlier results proofs can be found in the Aarhus lecture notes [1].

A word on related work. A general motivation has been the search for a form of generalized domain theory suitable for the semantics of concurrent processes and proofs [7]. An early definition of concurrent games appears in [8], where Samson Abramsky and Paul-André Melliès presented deterministic concurrent strategies as, essentially, partial closure operators on the domain of configurations of an event structure; such an operator takes any reachable configuration

of the game to the result of playing the intended moves of Player. Their motivation was the representation of proofs in linear logic. There followed a battery of insightful papers by Melliès and colleagues on *asynchronous games* culminating in the definition of *ingenuous strategies*—see *e.g.* [4,9]. The *receptive* ingenuous strategies of Melliès and Samuel Mimram have been shown to coincide with the deterministic concurrent strategies of [5], so justifying receptive ingenuous strategies as the most general deterministic concurrent strategies for which copycat behaves as identity. In comparison with early work of Abramsky and Martin Hyland on winning conditions in sequential games, the work here is closer to Hyland's, which it can be seen as extending [10,11]. The extension to games of imperfect information was guided solely by the wish to handle such games in a way that respected the bicategorical structure on concurrent games. There are however striking similarities with work by Abramsky and Radha Jagadeesan on games for access control [12].

2 Event Structures and Stable Families

An *event structure* comprises $(E, \mathrm{Con}, \leq)$, consisting of a set E, of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a nonempty *consistency relation* Con consisting of finite subsets of E, which satisfy

$$\begin{array}{l}\{e' \mid e' \leq e\} \text{ is finite for all } e \in E,\\ \{e\} \in \mathrm{Con} \text{ for all } e \in E,\\ Y \subseteq X \in \mathrm{Con} \implies Y \in \mathrm{Con}, \text{ and}\\ X \in \mathrm{Con} \;\&\; e \leq e' \in X \implies X \cup \{e\} \in \mathrm{Con}.\end{array}$$

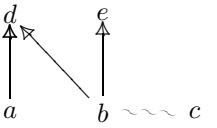
The *configurations*, $\mathcal{C}^\infty(E)$, of an event structure E consist of those subsets $x \subseteq E$ which are

Consistent: $\forall X \subseteq x.\ X \text{ is finite} \Rightarrow X \in \mathrm{Con}$, and
Down-closed: $\forall e, e'.\ e' \leq e \in x \implies e' \in x$.

Often we shall be concerned with just the finite configurations of an event structure. We write $\mathcal{C}(E)$ for the *finite* configurations of an event structure E.

Two events which are both consistent and incomparable w.r.t. causal dependency in an event structure are regarded as *concurrent*. In games the relation of *immediate* dependency $e \rightarrowtriangle e'$, meaning e and e' are distinct with $e \leq e'$ and no event in between, will play a very important role. For $X \subseteq E$ we write $[X]$ for $\{e \in E \mid \exists e' \in X.\ e \leq e'\}$, the down-closure of X; note if $X \in \mathrm{Con}$, then $[X] \in \mathrm{Con}$.

Example 1. The diagram below represents an event structure with five events in which, for example, d causally depends on the previous occurrence of a and b, while the two events b and c are inconsistent with each other (the squiggly line represents that the $\{b, c\}$ is not consistent).



As $\{b, c\}$ is not consistent neither is $\{e, c\}$, but we need not draw this as it is entailed by the axioms on the consistency relation. Often consistency/inconsistency is determined in a binary fashion and we can take advantage of this in a diagram of the event structure. However, this is not always the case. Consider for instance the event structure consisting of the three events $1, 2, 3$ with the discrete order and consistency relation

$$\mathrm{Con} = \{\ \emptyset,\ \{1\},\{2\},\{3\},\ \{1,2\},\{1,3\},\{2,3\}\ \}$$

□

Operations such as synchronized parallel composition are awkward to define directly on the simple event structures above. It is useful to broaden event structures to stable families, where operations are often carried out more easily, and then turned into event structures by the operation Pr below.

A *stable family* comprises $\mathcal{F}$, a nonempty family of finite subsets, called *configurations*, which satisfy:
Completeness: $\forall Z \subseteq \mathcal{F}.\ Z\uparrow \implies \bigcup Z \in \mathcal{F}$;
Coincidence-freeness: For all $x \in \mathcal{F}$, $e, e' \in x$ with $e \neq e'$,

$$\exists y \in \mathcal{F}.\ y \subseteq x\ \&\ (e \in y \iff e' \notin y)\,;$$

Stability: $\forall x, y \in \mathcal{F}.\ x \uparrow y \implies x \cap y \in \mathcal{F}$.
Above, $Z\uparrow$ means $\exists x \in \mathcal{F} \forall z \in Z.\ z \subseteq x$, and expresses the compatibility of Z in $\mathcal{F}$; we use $x \uparrow y$ for $\{x, y\}\uparrow$. We call elements of $\bigcup \mathcal{F}$ *events* of $\mathcal{F}$.

Proposition 1. *Let x be a configuration of a stable family $\mathcal{F}$. For $e, e' \in x$ define*

$$e' \leq_x e \text{ iff } \forall y \in \mathcal{F}.\ y \subseteq x\ \&\ e \in y \implies e' \in y.$$

When $e \in x$ define the prime configuration

$$[e]_x = \bigcap \{y \in \mathcal{F} \mid y \subseteq x\ \&\ e \in y\}\ .$$

Then $\leq_x$ is a partial order and $[e]_x$ is a configuration such that

$$[e]_x = \{e' \in x \mid e' \leq_x e\}.$$

Moreover the configurations $y \subseteq x$ are exactly the down-closed subsets of $\leq_x$.

Proposition 2. *Let $\mathcal{F}$ be a stable family. Then,* $\Pr(\mathcal{F}) =_{\mathrm{def}} (P, \mathrm{Con}, \leq)$ *is an event structure where:*

$$P = \{[e]_x \mid e \in x\ \&\ x \in \mathcal{F}\}\ ,$$
$$Z \in \mathrm{Con} \text{ iff } Z \subseteq P\ \&\ \bigcup Z \in \mathcal{F} \text{ and,}$$
$$p \leq p' \text{ iff } p, p' \in P\ \&\ p \subseteq p'\ .$$

A (partial) map of stable families $f : \mathcal{F} \to \mathcal{G}$ is a partial function f from the events of $\mathcal{F}$ to the events of $\mathcal{G}$ such that for all configurations $x \in \mathcal{F}$,

$$fx \in \mathcal{G} \;\&\; (\forall e_1, e_2 \in x.\ f(e_1) = f(e_2) \implies e_1 = e_2)\,.$$

Maps of event structures are maps of their stable families of configurations. Maps compose as functions. We say a map is *total* when it is total as a function.

Pr is the right adjoint of the "inclusion" functor, taking an event structure E to the stable family $\mathcal{C}(E)$. The unit of the adjunction $E \to \Pr(\mathcal{C}(E))$ takes an event e to the prime configuration $[e] =_{\text{def}} \{e' \in E \mid e' \leq e\}$. The counit $max : \mathcal{C}(\Pr(\mathcal{F})) \to \mathcal{F}$ takes prime configuration $[e]_x$ to its maximum event e; the image of a configuration $x \in \mathcal{C}(\Pr(\mathcal{F}))$ under the map max is $\bigcup x \in \mathcal{F}$.

Definition 1. Let $\mathcal{F}$ be a stable family. We use $x \mathrel{-\!\!\subset} y$ to mean y covers x in $\mathcal{F}$, *i.e.* $x \subset y$ in $\mathcal{F}$ with nothing in between, and $x \overset{e}{\mathrel{-\!\!\subset}} y$ to mean $x \cup \{e\} = y$ for $x, y \in \mathcal{F}$ and event $e \notin x$. We sometimes use $x \overset{e}{\mathrel{-\!\!\subset}}$, expressing that event e is enabled at configuration x, when $x \overset{e}{\mathrel{-\!\!\subset}} y$ for some y. W.r.t. $x \in \mathcal{F}$, write $[e)_x =_{\text{def}} \{e' \in E \mid e' \leq_x e \;\&\; e' \neq e\}$, so, for example, $[e)_x \overset{e}{\mathrel{-\!\!\subset}} [e]_x$. The relation of *immediate* dependence of event structures generalizes: with respect to $x \in \mathcal{F}$, the relation $e \rightarrowtriangle_x e'$ means $e \leq_x e'$ with $e \neq e'$ and no event in between.

3 Process Operations

3.1 Products

Let $\mathcal{A}$ and $\mathcal{B}$ be stable families with events A and B, respectively. Their product, the stable family $\mathcal{A} \times \mathcal{B}$, has events comprising pairs in $A \times_* B =_{\text{def}} \{(a, *) \mid a \in A\} \cup \{(a, b) \mid a \in A \;\&\; b \in B\} \cup \{(*, b) \mid b \in B\}$, the product of sets with partial functions, with (partial) projections π_1 and π_2—treating $*$ as 'undefined'—with configurations $x \in \mathcal{A} \times \mathcal{B}$ iff

$$\begin{array}{l} x \text{ is a finite subset of } A \times_* B \text{ s.t. } \pi_1 x \in \mathcal{A} \;\&\; \pi_2 x \in \mathcal{B}, \\ \forall e, e' \in x.\ \pi_1(e) = \pi_1(e') \text{ or } \pi_2(e) = \pi_2(e') \Rightarrow e = e'\,, \& \\ \forall e, e' \in x.\ e \neq e' \Rightarrow \exists y \subseteq x.\ \pi_1 y \in \mathcal{A} \;\&\; \pi_2 y \in \mathcal{B} \;\&\; (e \in y \iff e' \notin y)\,. \end{array}$$

Right adjoints preserve products. Consequently we obtain a product of event structures A and B by first regarding them as stable families $\mathcal{C}(A)$ and $\mathcal{C}(B)$, forming their product $\mathcal{C}(A) \times \mathcal{C}(B), \pi_1, \pi_2$, and then constructing the event structure

$$A \times B =_{\text{def}} \Pr(\mathcal{C}(A) \times \mathcal{C}(B))$$

and its projections as $\mathit{\Pi}_1 =_{\text{def}} \pi_1 max$ and $\mathit{\Pi}_2 =_{\text{def}} \pi_2 max$.

Later we shall use the following lemma relating immediate causal dependency in a product of stable families to immediate dependency in the components.

Lemma 1. *Suppose $e \rightarrowtriangle_x e'$ in a product of stable families $\mathcal{A} \times \mathcal{B}, \pi_1, \pi_2$.*
(i) If $e = (a, *)$ then $e' = (a', b)$ or $e' = (a', *)$ with $a \rightarrowtriangle_{\pi_1 x} a'$ in $\mathcal{A}$.
(ii) If $e' = (a', *)$ then $e = (a, b)$ or $e = (a, *)$ with $a \rightarrowtriangle_{\pi_1 x} a'$ in $\mathcal{A}$.
(iii) If $e = (a, b)$ and $e' = (a', b')$ then $a \rightarrowtriangle_{\pi_1 x} a'$ in $\mathcal{A}$ or $b \rightarrowtriangle_{\pi_2 x} b'$ in $\mathcal{B}$.

3.2 Restriction

The *restriction* of $\mathcal{F}$ to a subset of events R is the stable family $\mathcal{F} \restriction R =_{\text{def}} \{x \in \mathcal{F} \mid x \subseteq R\}$. Defining $E \restriction R$, the restriction of an event structure E to a subset of events R, to have events $E' = \{e \in E \mid [e] \subseteq R\}$ with causal dependency and consistency induced by E, we obtain $\mathcal{C}(E \restriction R) = \mathcal{C}(E) \restriction R$.

Proposition 3. *Let $\mathcal{F}$ be a stable family and R a subset of its events. Then,* $\Pr(\mathcal{F} \restriction R) = \Pr(\mathcal{F}) \restriction max^{-1} R$.

3.3 Synchronized Compositions

Synchronized parallel compositions are obtained as restrictions of products to those events which are allowed to synchronize or occur asynchronously according to the specific synchronized composition. For example, the synchronized composition of Milner's CCS on stable families $\mathcal{A}$ and $\mathcal{B}$ (with labelled events) is defined as $\mathcal{A} \times \mathcal{B} \restriction R$ where R comprises events which are pairs $(a, *)$, $(*, b)$ and (a, b), where in the latter case the events a of $\mathcal{A}$ and b of $\mathcal{B}$ carry complementary labels. Similarly, synchronized compositions of event structures A and B are obtained as restrictions $A \times B \restriction R$. By Proposition 3, we can equivalently form a synchronized composition of event structures by forming the synchronized composition of their stable families of configurations, and then obtaining the resulting event structure—this has the advantage of eliminating superfluous events earlier.

3.4 Projection

Event structures support a simple form of hiding. Let $(E, \leq, \text{Con})$ be an event structure. Let $V \subseteq E$ be a subset of 'visible' events. Define the *projection* of E on V, to be $E{\downarrow}V =_{\text{def}} (V, \leq_V, \text{Con}_V)$, where $v \leq_V v'$ iff $v \leq v'$ & $v, v' \in V$ and $X \in \text{Con}_V$ iff $X \in \text{Con}$ & $X \subseteq V$.

4 Event Structures with Polarities

We shall represent both a game and a strategy in a game as an event structure with polarity, which comprises (E, pol) where E is an event structure with a polarity function $pol : E \to \{+, -\}$ ascribing a polarity + (Player) or − (Opponent) to its events. The events correspond to (occurrences of) moves. Maps of event structures with polarity are maps of event structures which preserve polarity.

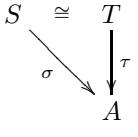
4.1 Operations

Dual. The *dual*, $E^{\perp}$, of an event structure with polarity E comprises a copy of the event structure E but with a reversal of polarities. It obviously extends to a functor. Write $\overline{e} \in E^{\perp}$ for the event complementary to $e \in E$ and *vice versa*.

Simple Parallel Composition. This operation simply juxtaposes two event structures with polarity. Let $(A, \leq_A, \mathrm{Con}_A, pol_A)$ and $(B, \leq_B, \mathrm{Con}_B, pol_B)$ be event structures with polarity. The events of $A \| B$ are $(\{1\} \times A) \cup (\{2\} \times B)$, their polarities unchanged, with: the only relations of causal dependency given by $(1, a) \leq (1, a')$ iff $a \leq_A a'$ and $(2, b) \leq (2, b')$ iff $b \leq_B b'$; a subset of events C is consistent in $A \| B$ iff $\{a \mid (1, a) \in C\} \in \mathrm{Con}_A$ and $\{b \mid (2, b) \in C\} \in \mathrm{Con}_B$. The operation extends to a functor—put the two maps in parallel. The empty event structure with polarity , written $\emptyset$, is the unit w.r.t. $\|$.

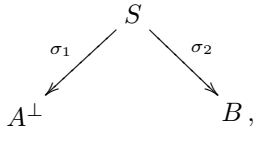
5 Pre-strategies

Let A be an event structure with polarity, thought of as a game; its events stand for the possible occurrences of moves of Player and Opponent and its causal dependency and consistency relations for the constraints imposed by the game. A *pre-strategy* in A is a total map $\sigma : S \to A$ from an event structure with polarity S. A pre-strategy represents a nondeterministic play of the game—all its moves are moves allowed by the game and obey the constraints of the game; the concept will later be refined to that of *strategy* (and *winning strategy* in Section 7). Two pre-strategies $\sigma : S \to A$ and $\tau : T \to A$ in A will be essentially the same when they are isomorphic, *i.e.* there is an isomorphism $S \cong T$ such that



commutes. Then we write $\sigma \cong \tau$.

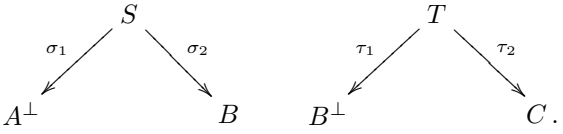
Let A and B be event structures with polarity. Following Joyal [13], a pre-strategy from A to B is a pre-strategy in $A^{\perp} \| B$, so a total map $\sigma : S \to A^{\perp} \| B$. It thus determines a span



of event structures with polarity where σ_1, σ_2 are *partial* maps. In fact, a pre-strategy from A to B corresponds to such spans where for all $s \in S$ either, but not both, $\sigma_1(s)$ or $\sigma_2(s)$ is defined. Two pre-strategies from A to B will be isomorphic when they are isomorphic as pre-strategies in $A^{\perp} \| B$, or equivalently are isomorphic as spans. We write $\sigma : A \mathrel{-\!\!\!\rightarrowtail} B$ to express that σ is a pre-strategy from A to B. Note a pre-strategy σ in a game A coincides with a pre-strategy from the empty game $\sigma : \emptyset \mathrel{-\!\!\!\rightarrowtail} A$.

5.1 Composing Pre-strategies

Consider two pre-strategies $\sigma : A \rightarrowtriangle B$ and $\tau : B \rightarrowtriangle C$ as spans:



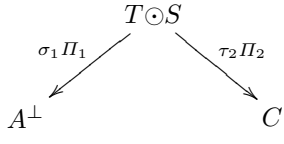
We show how to define their composition $\tau \odot \sigma : A \rightarrowtriangle C$ as the result of a synchronized composition, followed by projection to hide internal synchronization events. We first form the synchronized composition of S and T by restricting the product $S \times T$, with projections $\Pi_1 : S \times T \to S$ and $\Pi_2 : S \times T \to T$, to allow only those synchronizations associated with complementary events, of different polarities, in B and $B^\perp$. Specifically, the synchronized composition is $S \times T \restriction Syn$ where

$$\begin{aligned} Syn \;=\; & \{p \in S \times T \mid \sigma_1\Pi_1(p) \text{ is defined } \&\ \Pi_2(p) \text{ is undefined}\} \cup \\ & \{p \in S \times T \mid \tau_2\Pi_2(p) \text{ is defined } \&\ \Pi_1(p) \text{ is undefined}\} \cup \\ & \{p \in S \times T \mid \sigma_2\Pi_1(p) = \overline{\tau_1\Pi_2(p)} \text{ with both defined}\}\,. \end{aligned}$$

We define $T \odot S =_{\text{def}} (S \times T \restriction Syn) \downarrow V$ where

$$V = \{p \in S \times T \restriction Syn \mid \sigma_1\Pi_1(p) \text{ is defined}\} \cup \{p \in S \times T \restriction Syn \mid \tau_2\Pi_2(p) \text{ is defined}\}\,.$$

Finally, the composition $\tau \odot \sigma$ is defined to be the span



As remarked in Section 3.3, the same construction is achieved by first forming the synchronized composition of the stable families $\mathcal{C}(S)$ and $\mathcal{C}(T)$ (we often use this description in proofs):

Proposition 4. *The composition* $T \odot S = \Pr(\mathcal{C}(S) \times \mathcal{C}(T) \restriction R) \downarrow V$*, where*

$$\begin{aligned} R \;=\; & \{(s, *) \mid s \in S \ \&\ \sigma_1(s) \textit{ is defined}\} \cup \{(*, t) \mid t \in T \ \&\ \tau_2(t) \textit{ is defined}\} \cup \\ & \{(s, t) \mid s \in S \ \&\ t \in T \ \&\ \sigma_2(s) = \overline{\tau_1(t)} \textit{ with both defined}\}\,. \end{aligned}$$

The span $\tau \odot \sigma$ *comprises maps* $\upsilon_1 : T \odot S \to A^\perp$ *and* $\upsilon_2 : T \odot S \to C$*, which on events* p *of* $T \odot S$ *act so* $\upsilon_1(p) = \sigma_1(s)$ *when* $max(p) = (s, *)$ *and* $\upsilon_2(p) = \tau_2(t)$ *when* $max(p) = (*, t)$*, and are undefined elsewhere.*

The natural isomorphism $S \times (T \times U) \cong (S \times T) \times U$, associated with the product of event structures S, T, U, restricts to the required isomorphism of spans as the synchronizations involved in successive compositions are disjoint:

Proposition 5. *Let* $\sigma : A \rightarrowtriangle B$*,* $\tau : B \rightarrowtriangle C$ *and* $\upsilon : C \rightarrowtriangle D$ *be pre-strategies. The two compositions* $\upsilon \odot (\tau \odot \sigma)$ *and* $(\upsilon \odot \tau) \odot \sigma$ *are isomorphic.*

5.2 Concurrent Copy-Cat

Identities w.r.t. composition are given by copy-cat strategies. Let A be an event structure with polarity. The copy-cat strategy from A to A is an instance of a pre-strategy, so a total map $\gamma_A : \mathbb{CC}_A \to A^\perp \| A$. It describes a concurrent, or distributed, strategy based on the idea that Player moves, of $+$ve polarity, always copy previous corresponding moves of Opponent, of $-$ve polarity.

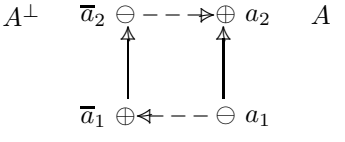
For $c \in A^\perp \| A$ we use $\overline{c}$ to mean the corresponding copy of c, of opposite polarity, in the alternative component, *i.e.* $\overline{(1,a)} = (2,\overline{a})$ and $\overline{(2,a)} = (1,\overline{a})$. Define $\mathbb{CC}_A$ to comprise the event structure with polarity $A^\perp \| A$ together with extra causal dependencies $\overline{c} \leq_{\mathbb{CC}_A} c$ for all events c with $pol_{A^\perp \| A}(c) = +$.

Proposition 6. *Let A be an event structure with polarity. Then $\mathbb{CC}_A$ is an event structure with polarity. Moreover,*

$$x \in \mathcal{C}(\mathbb{CC}_A) \text{ iff } x \in \mathcal{C}(A^\perp \| A) \ \&\ \forall c \in x.\ pol_{A^\perp \| A}(c) = + \implies \overline{c} \in x\,.$$

The *copy-cat* pre-strategy $\gamma_A : A \rightarrowtail\!\!\!\!\!\rightarrow A$ is defined to be the map $\gamma_A : \mathbb{CC}_A \to A^\perp \| A$ where γ_A is the identity on the common set of events.

Example 2. We illustrate the construction of the copy-cat strategy for the event structure A comprising the single immediate dependency $a_1 \rightarrowtriangle a_2$ from an Opponent move a_1 to a Player move a_2. The event structure $\mathbb{CC}_A$ is obtained from $A^\perp \| A$ by adjoining the additional immediate dependencies shown:



The pre-strategy $\gamma_A : \mathbb{CC}_A \to A^\perp \| A$ is defined to act simply as the identity function on events. □

6 Strategies

The main result of [5], presented summarily here, is that two conditions on pre-strategies, *receptivity* and *innocence*, are necessary and sufficient for copy-cat to behave as identity w.r.t. the composition of pre-strategies. Receptivity ensures an openness to all possible moves of Opponent. Innocence restricts the behaviour of Player; Player may only introduce new relations of immediate causality of the form $\ominus \rightarrowtriangle \oplus$ beyond those imposed by the game.

Receptivity. A pre-strategy σ is *receptive* iff $\sigma x \overset{a}{\text{—}\!\!\subset}$ & $pol_A(a) = - \Rightarrow \exists! s \in S.\ x \overset{s}{\text{—}\!\!\subset}$ & $\sigma(s) = a$.

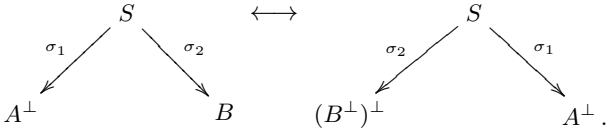
Innocence. A pre-strategy σ is *innocent* when it is both
$+$*-innocent:* if $s \rightarrowtriangle s'$ & $pol(s) = +$ then $\sigma(s) \rightarrowtriangle \sigma(s')$, and
$-$*-innocent:* if $s \rightarrowtriangle s'$ & $pol(s') = -$ then $\sigma(s) \rightarrowtriangle \sigma(s')$.

Theorem 1. *Let $\sigma : A \mathrel{-\!\!\!\rightarrowtriangle} B$ be pre-strategy. Copy-cat behaves as identity w.r.t. composition, i.e. $\sigma \circ \gamma_A \cong \sigma$ and $\gamma_B \circ \sigma \cong \sigma$, iff σ is receptive and innocent. Copy-cat pre-stategies $\gamma_A : A \mathrel{-\!\!\!\rightarrowtriangle} A$ are receptive and innocent.*

6.1 The Bicategory of Concurrent Games and Strategies

Theorem 1 motivates the definition of a *strategy* as a pre-strategy which is receptive and innocent. In fact, we obtain a bicategory, **Games**, in which the objects are event structures with polarity—the games, the arrows from A to B are strategies $\sigma : A \mathrel{-\!\!\!\rightarrowtriangle} B$ and the 2-cells are maps of spans. The vertical composition of 2-cells is the usual composition of maps of spans. Horizontal composition is given by the composition of strategies $\odot$ (which extends to a functor on 2-cells via the functoriality of synchronized composition).

A strategy $\sigma : A \mathrel{-\!\!\!\rightarrowtriangle} B$ corresponds to a dual strategy $\sigma^\perp : B^\perp \mathrel{-\!\!\!\rightarrowtriangle} A^\perp$. This duality arises from the correspondence



The dual of copy-cat, $\gamma_A^\perp$, is isomorphic to the copy-cat of the dual, $\gamma_{A^\perp}$, for A an event structure with polarity. The dual of a composition of pre-strategies $(\tau \odot \sigma)^\perp$ is isomorphic to the composition $\sigma^\perp \odot \tau^\perp$. This duality will be maintained in all the bicategories of games we shall consider.

6.2 The Subcategory of Deterministic Strategies

Say an event structure with polarity S is *deterministic* iff

$$\forall X \subseteq_{\text{fin}} S.\ Neg[X] \in \text{Con}_S \implies X \in \text{Con}_S\,,$$

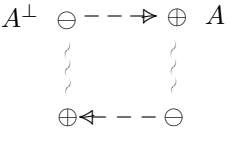
where $Neg[X] =_{\text{def}} \{s' \in S \mid pol(s') = - \ \&\ \exists s \in X.\ s' \leq s\}$. In other words, S is deterministic iff any finite set of moves is consistent when it causally depends only on a consistent set of opponent moves. Say a strategy $\sigma : S \to A$ is deterministic if S is deterministic.

Lemma 2. *An event structure with polarity S is deterministic iff*

$$\forall s, s' \in S, x \in \mathcal{C}(S).\ \ x \overset{s}{\text{—}\!\subset}\ \&\ x \overset{s'}{\text{—}\!\subset}\ \&\ pol(s) = + \implies x \cup \{s, s'\} \in \mathcal{C}(S)\,.$$

In general, a copy-cat strategy can fail to be deterministic, illustrated below.

Example 3. Take A to consist of two events, one $+$ve and one $-$ve event, inconsistent with each other. The construction $\mathbb{C}_A$:



To see $\mathbb{C}_A$ is not deterministic, take x to be the singleton set consisting *e.g.* of the $-$ve event on the left and s, s' to be the $+$ve and $-$ve events on the right. □

Copy-cat γ_A is deterministic iff immediate conflict in A respects polarity, or equivalently that there is no immediate conflict between $+$ve and $-$ve events, a condition we call 'race-free.'

Lemma 3. *Let A be an event structure with polarity. The copy-cat strategy γ_A is deterministic iff*

$$\forall x \in \mathcal{C}(A).\ \ x \overset{a}{\text{—}\subset}\ \&\ x \overset{a'}{\text{—}\subset}\ \&\ pol(a) = +\ \&\ pol(a') = - \implies x \cup \{a, a'\} \in \mathcal{C}(A)\,. \qquad (\mathbf{Race-free})$$

Lemma 4. *The composition of deterministic strategies is deterministic.*

Lemma 5. *A deterministic strategy $\sigma : S \to A$ is injective on configurations (equivalently, σ is mono in the category of event structures with polarity).*

We obtain a sub-bicategory **DGames** of **Games** by restricting objects to race-free games and strategies to being deterministic. Via Lemma 5, deterministic strategies in a game correspond to certain subfamilies of configurations of the game. A characterization of those subfamilies which correspond to deterministic strategies shows them to coincide with the receptive ingenuous strategies of Mimram and Melliès [9]. Via the presentation of deterministic strategies as families **DGames** is equivalent to an order-enriched category.

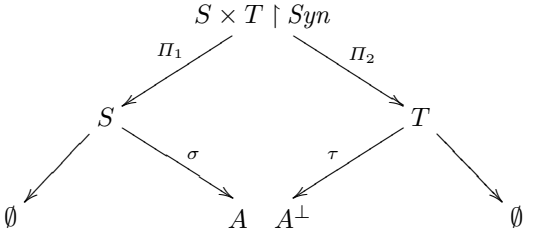
7 Winning, Losing and Drawing

A *game with winning/losing conditions* comprises $G = (A, W, L)$ where A is an event structure with polarity and $W \subseteq \mathcal{C}^\infty(A)$ consists of the *winning* configurations disjoint from the *losing* configurations $L \subseteq \mathcal{C}^\infty(A)$ for Player. We do not insist that W and L partition the set $\mathcal{C}^\infty(A)$—there may be *neutral* configurations at which Player and Opponent draw.

A strategy in G is a strategy in A. A strategy in G is regarded as *winning* if it always prescribes Player moves to end up in a winning configuration, no matter what the activity or inactivity of Opponent. Formally, a strategy $\sigma : S \to A$ in G is *winning (for Player)* if $\sigma x \in W$ for all $+$-maximal configurations $x \in \mathcal{C}^\infty(S)$—a configuration x is $+$-maximal if whenever $x \overset{s}{\text{—}\subset}$ then the event s has $-$ve polarity. Any achievable position $z \in \mathcal{C}^\infty(S)$ of the game can be extended to a $+$-maximal, so winning, configuration (via Zorn's Lemma). So a strategy prescribes Player moves to reach a winning configuration whatever state of play is achieved following the strategy. Note that for a game A, if winning conditions $W = \mathcal{C}^\infty(A)$, *i.e.* every configuration is winning, then any strategy in A is a winning strategy.

Informally, we can also understand a strategy as winning for Player if when played against any counter-strategy of Opponent, the final result is a win for

Player. Suppose $\sigma : S \to A$ is a strategy in a game (A, W). A counter-strategy is strategy of Opponent, so a strategy $\tau : T \to A^{\perp}$ in the dual game. We can view σ as a strategy $\sigma : \emptyset \multimap A$ and τ as a strategy $\tau : A \multimap \emptyset$. Their composition $\tau \odot \sigma : \emptyset \multimap \emptyset$ is not in itself so informative. Rather it is the status of the configurations in $\mathcal{C}^{\infty}(A)$ their full interaction induces which decides which of Player or Opponent wins. For this we should consider the composition of σ and τ *before* hiding internal synchronizations:



where

$$Syn \;=\; \{p \in S \times T \mid \sigma\Pi_1(p) = \overline{\tau\Pi_2(p)} \text{ with both defined}\}\,.$$

Because σ or τ may be nondeterministic there can be more than one maximal configuration z in $\mathcal{C}^{\infty}(S \times T \restriction Syn)$. A maximal configuration z in $\mathcal{C}^{\infty}(S \times T \restriction Syn)$ images to a configuration $\sigma\Pi_1 z = \tau\Pi_2 z$ in $\mathcal{C}^{\infty}(A)$. Define the set of *results* of the interaction of σ and τ to be

$$\langle \sigma, \tau \rangle =_{\text{def}} \{\sigma\Pi_1 z \mid z \text{ is maximal in } \mathcal{C}^{\infty}(S \times T \restriction Syn)\}\,.$$

A configuration $x \in \langle \sigma, \tau \rangle$, resulting from a play of σ against τ may be a win for Player, if $x \in W$, a loss for Player and a win for Opponent, if $x \in L$, or a draw, when $x \notin W \cup L$.

Lemma 6. *Let $\sigma : S \to A$ be a strategy in a game (A, W, L). The strategy σ is a winning for Player iff $\langle \sigma, \tau \rangle \subseteq W$ for all (deterministic) strategies $\tau : T \to A^{\perp}$.*

Corollary 1. *There are the following three equivalent ways to say that a strategy $\sigma : S \to A$ is winning in (A, W, L):*

1. *$\sigma x \in W$ for all +-maximal configurations $x \in \mathcal{C}^{\infty}(S)$, i.e. the strategy prescribes Player moves to reach a winning configuration, no matter what the activity or inactivity of Opponent;*
2. *$\langle \sigma, \tau \rangle \subseteq W$ for all strategies $\tau : T \to A^{\perp}$, i.e. all plays against counter-strategies of the Opponent result in a win for Player;*
3. *$\langle \sigma, \tau \rangle \subseteq W$ for all deterministic strategies $\tau : T \to A^{\perp}$, i.e. all plays against deterministic counter-strategies of the Opponent result in a win for Player.*

The proof of Lemma 6 relies on the following general lemma which will be useful later. Its proof and those of the lemma and corollary above can be found in [1]—they are essentially repeats of the corresponding proofs for games with just winning conditions.

Lemma 7. *Let $\sigma : S \to A^\perp \| B$ and $\tau : T \to B^\perp \| C$ be receptive pre-strategies. Then,*

$$z \in \mathcal{C}^\infty(T \times S \restriction Syn) \text{ is } +\text{-maximal iff}$$
$$\Pi_1 z \in \mathcal{C}^\infty(S) \text{ is } +\text{-maximal } \& \ \Pi_2 z \in \mathcal{C}^\infty(T) \text{ is } +\text{-maximal.}$$

A convention is being adopted in Lemma 7. The events of $T \times S \restriction Syn$ are constructed as prime configurations p of a stable family, and as such $max(p)$ has the form $(s, *)$, $(*, t)$ or (s, t), with $s \in S$ and $t \in T$. An event p with $max(p)$ of the form $(s, *)$ or $(*, t)$ adopts the polarity of the event s or t, while those p with $max(p) = (s, t)$ are regarded as not having a polarity. By $x \in \mathcal{C}^\infty(T \times S \restriction Syn)$ is $+$-maximal is meant that whenever $x \overset{e}{\text{—}\!\subset}$ the event e has $-$ve polarity.

8 Operations

8.1 Dual

There is an obvious dual of a game $G = (A, W_G, L_G)$ which reverses the role of Player and Opponent: $G^\perp = (A^\perp, W_{G^\perp}, L_{G^\perp})$ where

$$x \in W_{G^\perp} \iff \overline{x} \in L_G \text{ and}$$
$$x \in L_{G^\perp} \iff \overline{x} \in W_G\,.$$

Here, and in future, we extend the bar-notation for the bijection between events of A and $A^\perp$ to configurations: a configuration $x \in \mathcal{C}^\infty(A)$ corresponds to a configuration $\overline{x} =_{\text{def}} \{\overline{a} \mid a \in x\} \in \mathcal{C}^\infty(A^\perp)$.

8.2 Parallel Composition

The parallel composition of two games $G = (A, W_G, L_G)$, $H = (B, W_H, L_H)$ is

$$G \| H =_{\text{def}} (A \| B,\ W_G \| \mathcal{C}^\infty(B) \cup \mathcal{C}^\infty(A) \| W_H,\ L_G \| L_H)$$

where $X \| Y = \{\{1\} \times x \cup \{2\} \times y \mid x \in X \ \& \ y \in Y\}$ when X and Y are subsets of configurations. In other words, for $x \in \mathcal{C}^\infty(A \| B)$,

$$x \in W_{G\|H} \iff x_1 \in W_G \text{ or } x_2 \in W_H\,, \text{ and}$$
$$x \in L_{G\|H} \iff x_1 \in L_G \ \& \ x_2 \in L_H\,,$$

where $x_1 = \{a \mid (1, a) \in x\}$ and $x_2 = \{b \mid (2, b) \in x\}$. To win in $G \| H$ is to win in either game; to lose is to lose in both games. The unit of $\|$ is $(\emptyset, \emptyset, \{\emptyset\})$.

8.3 Tensor

For games $G = (A, W_G, L_G)$, $H = (B, W_H, L_H)$, defining $G \otimes H =_{\text{def}} (G^\perp \| H^\perp)^\perp$ we obtain a game where to win is to win in both games G and H, and to lose is to lose in either game. More explicitly,

$$(A, W_G, L_G) \otimes (B, W_H, L_H) =_{\text{def}} (A \| B,\ W_G \| W_H,\ L_G \| \mathcal{C}^\infty(B) \cup \mathcal{C}^\infty(A) \| L_H)\,.$$

The unit of $\otimes$ is $(\emptyset, \{\emptyset\}, \emptyset)$.

8.4 Function Space

With $G \multimap H =_{\text{def}} G^{\perp} \| H$ a win in $G \multimap H$ is a win in H conditional on not losing in G:

Proposition 1. *Let $G = (A, W_G, L_G)$ and $H = (B, W_H, L_H)$ be games with winning conditions. Write $W_{G \multimap H}$, $L_{G \multimap H}$ for the winning, respectively losing conditions of $G \multimap H$. For $x \in \mathcal{C}^{\infty}(A^{\perp} \| B)$,*

$$x \in W_{G \multimap H} \quad \textit{iff} \quad \overline{x_1} \notin L_G \implies x_2 \in W_H \,, \ \textit{and}$$
$$x \in L_{G \multimap H} \quad \textit{iff} \quad x_2 \notin L_H \implies \overline{x_1} \in W_G \,.$$

Proof. Letting $x \in \mathcal{C}^{\infty}(A^{\perp} \| B)$,

$$\begin{aligned} x \in W_{G \multimap H} \text{ iff } & \ x \in W_{G^{\perp} \| H} \\ \text{iff } & \ x_1 \in W_{G^{\perp}} \text{ or } x_2 \in W_H \\ \text{iff } & \ \overline{x_1} \in L_G \text{ or } x_2 \in W_H \\ \text{iff } & \ \overline{x_1} \notin L_G \implies x_2 \in W_H \,. \end{aligned}$$

The other part is proved similarly. □

9 The Bicategory of Winning Strategies

We can again follow Joyal and define strategies between games now with winning/losing conditions: a (winning) strategy from G, a game with winning/losing conditions, to another H is a (winning) strategy in $G \multimap H = G^{\perp} \| H$. We compose strategies as before. We show that the composition of winning strategies is winning.

Lemma 8. *Let σ be a winning strategy in $G^{\perp} \| H$ and τ be a winning strategy in $H^{\perp} \| K$. Their composition $\tau \odot \sigma$ is a winning strategy in $G^{\perp} \| K$.*

Proof. Suppose $x \in \mathcal{C}^{\infty}(T \odot S)$ is $+$-maximal. The event structure $T \odot S$ is obtained as the projection of $S \times T \upharpoonright \mathit{Syn}$ to the set of 'visible' events V. Hence the down-closure $[x]$ in $S \times T \upharpoonright \mathit{Syn}$ forms a configuration $[x] \in \mathcal{C}^{\infty}(S \times T \upharpoonright \mathit{Syn})$. By Zorn's Lemma we can extend $[x]$ to a maximal configuration $z \supseteq [x]$ in $\mathcal{C}^{\infty}(S \times T \upharpoonright \mathit{Syn})$ with the property that all events of $z \setminus [x]$ are synchronizations of the form p with $\mathit{max}(p) = (s, t)$ for $s \in S$ and $t \in T$. Then, z will be $+$-maximal in $\mathcal{C}^{\infty}(S \times T \upharpoonright \mathit{Syn})$ with

$$\sigma_1 \Pi_1 z = \sigma_1 \Pi_1 [x] \quad \& \quad \tau_2 \Pi_2 z = \tau_2 \Pi_2 [x] \,. \tag{1}$$

By Lemma 7,

$$\Pi_1 z \text{ is } + \text{-maximal in } S \quad \& \quad \Pi_2 z \text{ is } + \text{-maximal in } T \,.$$

As σ and τ are winning,

$$\sigma \Pi_1 z \in W_{G^{\perp} \| H} \quad \& \quad \tau \Pi_2 z \in W_{H^{\perp} \| K} \,.$$

Now $\sigma \Pi_1 z \in W_{G^\perp \| H}$ expresses that

$$\overline{\sigma_1 \Pi_1 z} \notin L_G \implies \sigma_2 \Pi_1 z \in W_H \tag{2}$$

and $\tau \Pi_2 z \in W_{H^\perp \| K}$ that

$$\overline{\tau_1 \Pi_2 z} \notin L_H \implies \tau_2 \Pi_2 z \in W_K\,, \tag{3}$$

by Proposition 1. But

$$\sigma_2 \Pi_1 z \in W_H \implies \sigma_2 \Pi_1 z \notin L_H \tag{4}$$

as W_H and L_H are disjoint. Moreover, $\sigma_2 \Pi_1 z = \overline{\tau_1 \Pi_2 z}$. So (2), (3) and (4) yield

$$\overline{\sigma_1 \Pi_1 z} \notin L_G \implies \tau_2 \Pi_2 z \in W_K\,.$$

By (1)

$$\overline{\sigma_1 \Pi_1 [x]} \notin L_G \implies \tau_2 \Pi_2 [x] \in W_K\,,$$

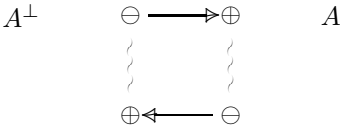
i.e. from the definition of $\tau \odot \sigma$,

$$\overline{(\tau \odot \sigma)_1\, x} \notin L_G \implies (\tau \odot \sigma)_2\, x \in W_K$$

in the span of the composition $\tau \odot \sigma$. Hence $\tau \odot \sigma\, x \in W_{G^\perp \| K}$ whenever x is a $+$-maximal configuration of $T \odot S$, ensuring $\tau \odot \sigma$ is a winning strategy. □

For a general game with winning/losing conditions (A, W, L) the copy-cat strategy need not be winning:

Example 4. Let A consist of two events, one $+$ve event $\oplus$ and one $-$ve event $\ominus$, inconsistent with each other. Take as winning conditions the set $W = \{\{\oplus\}\}$ and as losing conditions the set $L = \{\{\ominus\}\}$. The event structure $\mathbb{CC}_A$:



To see $\mathbb{CC}_A$ is not winning consider the configuration x consisting of the two $-$ve events in $\mathbb{CC}_A$. Then x is $+$-maximal as any $+$ve event is inconsistent with x. However, $\overline{x}_1 \notin L$ while $x_2 \notin W$, failing the winning conditions of $(A, W, L) \multimap (A, W, L)$. □

Each event structure with polarity$A$ possesses a 'Scott order' on its configurations $\mathcal{C}^\infty(A)$:

$$x' \sqsubseteq x \;\text{ iff }\; x' \supseteq^- x \cap x' \subseteq^+ x\,.$$

Above we use the special inclusions

$$x \subseteq^- y \text{ iff } x \subseteq y \;\&\; pol_A(y \setminus x) \subseteq \{-\}\,, \text{ and}$$
$$x \subseteq^+ y \text{ iff } x \subseteq y \;\&\; pol_A(y \setminus x) \subseteq \{+\}$$

for $x, y \in \mathcal{C}^\infty(A)$. A necessary and sufficient for copy-cat to be winning w.r.t. a game (A, W, L):

$$\forall x, x' \in \mathcal{C}^\infty(A).\ \begin{array}{l}\text{if } x' \sqsubseteq x \ \&\ x' \text{ is } +\text{-maximal } \&\ x \text{ is } -\text{-maximal,}\\ \text{then } \ x \in L \text{ or } x' \in W\,.\end{array} \qquad \textbf{(Cwins)}$$

Lemma 9. *Let (A, W, L) be a game with winning/losing conditions. The copy-cat strategy $\gamma_A : \mathbb{CC}_A \to A^\perp \| A$ is winning iff (A, W, L) satisfies* **(Cwins)**.

Proof. It can be shown that

$$z \in \mathcal{C}^\infty(\mathbb{CC}_A) \ \text{ iff } \ z = \{1\} \times \overline{x} \cup \{2\} \times x' \ \text{ with } \ x' \sqsubseteq_A x\,,$$

for $x, x' \in \mathcal{C}^\infty(A)$—see Lemma 54 in the Aarhus notes [1]. In this situation z is $+$-maximal iff both x is $-$-maximal and x' is $+$-maximal. Thus **(Cwins)** expresses precisely that copy-cat is winning in $(A, W, L) \multimap (A, W, L)$. □

For race-free games we can simplify (**Cwins**), the condition for copy-cat to be winning. Copy-cat is a winning strategy for a race-free game iff no maximal configuration of the game is neutral.

Proposition 2. *Assume A is a race-free event structure with polarity. For a game (A, W, L), the property* (**Cwins**) *holds iff $x \in W \cup L$ for all maximal configurations $x \in \mathcal{C}^\infty(A)$.*

Proof. For $x, x' \in \mathcal{C}^\infty(A)$, assume

$$x' \sqsubseteq x \ \&\ x' \text{ is } +\text{-maximal } \&\ x \text{ is } -\text{-maximal.}$$

As $x' \supseteq^- x \cap x' \subseteq^+ x$, there are covering chains associated with purely $+$ve and $-$ve events from $x \cap x'$ to x and x', respectively:

$$x \cap x' \ \overset{+}{\text{—}\!\subset}\ \cdots \ \overset{+}{\text{—}\!\subset}\ x\,,$$
$$x \cap x' \ \overset{-}{\text{—}\!\subset}\ \cdots \ \overset{-}{\text{—}\!\subset}\ x'\,.$$

If one of the covering chains is of zero length then so must the other be—otherwise we contradict one or other of the maximality assumptions. On the other hand, if both are nonempty, by repeated use of **(Race-free)** we again contradict a maximality assumption, *e.g.*

$$\begin{array}{ccccccc} y_1 & \overset{+}{\text{—}\!\subset} & x_1 \cup x_1' & \overset{+}{\text{—}\!\subset} & \cdots & \overset{+}{\text{—}\!\subset} & x \cup x_1' \\ -\big\uparrow & & -\big\uparrow & & & & -\big\uparrow \\ x \cap x' & \overset{+}{\text{—}\!\subset} & x_1 & \overset{+}{\text{—}\!\subset} & \cdots & \overset{+}{\text{—}\!\subset} & x \end{array}$$

shows how a repeated use of **(Race-free)** contradicts the $-$-maximality of x. We conclude that both covering chains must be of zero length, making $x = x \cap x' = x'$. As configurations which are both $+$ and $-$-maximal are simply maximal, the property **(Cwins)** now expresses that all maximal configurations are either winning or losing. □

We can now refine the bicategory of strategies **Games** to the bicategory **WLGames** with objects $G, H, \cdots$, games with winning/losing conditions satisfying (**Cwins**), and arrows winning strategies $G \rightarrowtail\!\!\!\!\blacktriangleright H$; 2-cells, their vertical and horizontal composition is as before. Via the constructions of Section 8, the bicategory is rich in categorical structure, and is in particular monoidal-closed. Its restriction to deterministic strategies yields a bicategory equivalent to a simpler order-enriched category.

10 Games with Imperfect Information

Consider the game "rock, scissors, paper" in which the two participants Player and Opponent independently sign one of r ("rock"), s ("scissors") or p ("paper"). The participant with the dominant sign w.r.t. the relation

$$r \textit{ beats } s,\ s \textit{ beats } p \text{ and } p \textit{ beats } r$$

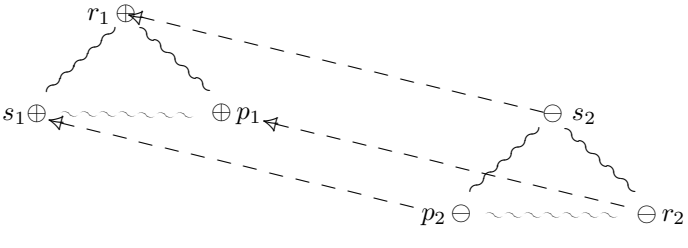
wins. It seems sensible to represent this game by RSP, the event structure with polarity

comprising the three mutually inconsistent possible signings of Player in parallel with the three mutually inconsistent signings of Opponent. What are the winning/losing conditions? A reasonable choice is to take the winning and losing configurations (for Player) to be given by

$$W = \{\{r_1, s_2\}, \{s_1, p_2\}, \{p_1, r_2\}\} \text{ and } L = \{\{s_1, r_2\}, \{p_1, s_2\}, \{r_1, p_2\}\}\,.$$

All other configurations are neutral, neither winning nor losing.

In this game it turns out that no participant has a winning strategy, which agrees with our informal understanding of the game "rock, scissors, paper." However on closer inspection there is a mismatch between the possible strategies allowed in our idealised mathematical and those of the real game. To make the mismatch clearer, let us bias the game in favour of Player by making the empty configuration winning, *i.e.* now $\emptyset \in W$. In this case there is a winning strategy for Player, *viz.* await the move of Opponent and then beat it with a dominant move. Explicitly, the winning strategy $\sigma : S \to RSP$ is given as the obvious map from S, the following event structure with polarity:



But this strategy hardly enters into the spirit of "rock, scissors, paper" where the participants are intended to make their moves *independently*. The problem with the game RSP as it stands is that it is a game of *perfect information* in the sense that all moves are visible to both participants. This permits the winning strategy above with its unwanted dependencies on moves which should be unseen by Player. To adequately model "rock, scissors, paper" requires a game of *imperfect information* where some moves are masked, or inaccessible, and strategies with dependencies on unseen moves are ruled out.

We extend concurrent games to games with imperfect information. To do so in way that respects the operations of the bicategory of games we suppose a fixed preorder of *levels* $(\Lambda, \preceq)$. The levels are to be thought of as levels of access, or permission. Moves in games and strategies are to respect levels: moves will be assigned levels in such a way that a move is only permitted to causally depend on moves at equal or lower levels; it is as if from a level only moves of equal or lower level can be seen.

An Λ-game (G, l) comprises a game $G = (A, W, L)$ with winning/losing conditions together with a *level function* $l : A \to \Lambda$ such that

$$a \leq_A a' \implies l(a) \preceq l(a')$$

for all $a, a' \in A$. A Λ-strategy in the Λ-game (G, l) is a strategy $\sigma : S \to A$ for which

$$s \leq_S s' \implies l\sigma(s) \preceq l\sigma(s')$$

for all $s, s' \in S$.

For example, for "rock, scissors, paper" we can take Λ to be the discrete preorder consisting of levels 1 and 2 unrelated to each other under $\preceq$. To make RSP into a suitable Λ-game the level function l takes +ve events in RSP to level 1 and −ve events to level 2. The strategy above, where Player awaits the move of Opponent then beats it with a dominant move, is now disallowed because it is not a Λ-strategy—it introduces causal dependencies which do not respect levels. If instead we took Λ to be the unique preorder on a single level the Λ-strategies would coincide with all the strategies.

Games with imperfect information are central to the semantics of Hintikka and Sandu's independence-friendly (IF) logic in which special quantifiers restrict those strategies permitted to establish an assertion [14]. A recent paper [15], building on a concurrent-game semantics of predicate calculus [6], proposes a compositional semantics for a variant of IF logic in which assertions of IF logic denote concurrent games with imperfect information.

10.1 The Bicategory of Λ-Games

The introduction of levels meshes smoothly with the bicategorical structure on games.

For a Λ-game (G, l_G), define its dual $(G, l_G)^\perp$ to be $(G^\perp, l_{G^\perp})$ where $l_{G^\perp}(\overline{a}) = l_G(a)$, for a an event of G.

For Λ-games (G, l_G) and (H, l_H), define their parallel composition $(G, l_G)\|(H, l_H)$ to be $(G\|H, l_{G\|H})$ where $l_{G\|H}((1, a)) = l_G(a)$, for a an event of G, and $l_{G\|H}((2, b)) = l_H(b)$, for b an event of H.

A strategy between Λ-games from (G, l_G) to (H, l_H) is a strategy in $(G, l_G)^\perp\|(H, l_H)$.

Proposition 3
(i) Let (G, l_G) be a Λ-game where G satisfies **(Cwins)**. *The copy-cat strategy on G is a Λ-strategy.*
(ii) The composition of Λ-strategies is a Λ-strategy.

Proof. (i) The additional causal links introduced in the construction of the copy-cat strategy are between complementary events in $G^\perp$ and G, at the same level in Λ, and so respect $\preceq$.

(ii) Let (G, l_G), (H, l_H) and (K, l_K) be Λ-games. Let $\sigma : G \mapsto\!\!\!\!\rightarrow H$ and $\tau : H \mapsto\!\!\!\!\rightarrow K$ be Λ-strategies. We show their composition $\tau\odot\sigma$ is a Λ-strategy.

It suffices to show $p \rightarrow p'$ in $T\odot S$ implies $l_{G^\perp\|K}\tau\odot\sigma(p) \preceq l_{G^\perp\|K}\tau\odot\sigma(p')$. Suppose $p \rightarrow p'$ in $T\odot S$ with $max(p) = e$ and $max(p') = e'$. Take $x \in \mathcal{C}(T\odot S)$ containing p' so p too. Then, referring to Proposition 4,

$$e \rightarrow\!\!\!\!\rightarrow_{\bigcup x} e_1 \rightarrow\!\!\!\!\rightarrow_{\bigcup x} \cdots \rightarrow\!\!\!\!\rightarrow_{\bigcup x} e_{n-1} \rightarrow\!\!\!\!\rightarrow_{\bigcup x} e'$$

where $e, e' \in V_0$ and $e_i \notin V_0$ for $1 \leq i \leq n-1$. (V_0 consists of 'visible' events of the stable family, those of the form $(s, *)$ with $\sigma_1(s)$ defined, or $(*, t)$, with $\tau_2(t)$ defined.) The events e_i have the form (s_i, t_i) where $\sigma_2(s_i) = \tau_1(t_i)$, for $1 \leq i \leq n-1$.

Any individual link in the chain above has one of the forms:

$$(s, t) \rightarrow\!\!\!\!\rightarrow_{\bigcup x} (s', t'),\ (s, *) \rightarrow\!\!\!\!\rightarrow_{\bigcup x} (s', t'),$$
$$(*, t) \rightarrow\!\!\!\!\rightarrow_{\bigcup x} (s', t'),\ (s, t) \rightarrow\!\!\!\!\rightarrow_{\bigcup x} (s', *),\ \text{ or } \ (s, t) \rightarrow\!\!\!\!\rightarrow_{\bigcup x} (*, t').$$

By Lemma 6, for any link either $s \rightarrow\!\!\!\!\rightarrow_S s'$ or $t \rightarrow\!\!\!\!\rightarrow_T t'$. As σ and τ are Λ-strategies, this entails

$$l_{G^\perp\|H}\sigma(s) \preceq l_{G^\perp\|H}\sigma(s') \ \text{ or } \ l_{H^\perp\|K}\tau(t) \preceq l_{H^\perp\|K}\tau(t')$$

for any link. Consequently $\preceq$ is respected across the chain and $l_{G^\perp\|K}\tau\odot\sigma(p) \preceq l_{G^\perp\|K}\tau\odot\sigma(p')$, as required. □

W.r.t. a particular choice of access levels $(\Lambda, \preceq)$ we obtain a bicategory $\mathbf{WLGames}_\Lambda$. Its objects are Λ-games (G, l) where G satisfies **(Cwins)** with arrows the Λ-strategies and 2-cells maps of spans. It restricts to a sub-bicategory of deterministic Λ-strategies, which as before is equivalent to an order-enriched category.

We can shift between different access levels. Let $r : (\Lambda, \preceq) \to (\Lambda', \preceq')$ be a monotonic function between preorders of levels. By composition with r a Λ-game (G, l) becomes a Λ'-game $(G, r \circ l)$, giving rise to a (pseudo) functor from $\mathbf{WGames}_\Lambda$ to $\mathbf{WGames}_{\Lambda'}$. Provided r is injective, the functor has a right adjoint from $\mathbf{WGames}_{\Lambda'}$ to $\mathbf{WGames}_\Lambda$.

Acknowledgments. Congratulations and thanks to Dexter Kozen for his dexterity across the areas of algorithmics and semantics. I'm grateful to the students at Aarhus who attended my course in the summer 2011, in particular to Nikoline Valsgaard Vinkel who bravely tackled the project on games with neutral positions. Thanks to my coworkers on the project ECSYM, Pierre Clairambault and Julian Gutierrez who in particular assisted in the Aarhus course. Thanks too to the anonymous referees for their helpful comments. The support of Advanced Grant ECSYM of the European Research Council is acknowledged with gratitude.

References

1. Winskel, G.: Event structures, stable families and games. Lecture notes, Comp Science Dept, Aarhus University (2011), `http://daimi.au.dk/~gwinskel`
2. Conway, J.: On Numbers and Games. A K Peters, Wellesley (2000)
3. Winskel, G.: Event Structure Semantics for CCS and Related Languages. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140. Springer, Heidelberg (1982)
4. Melliès, P.A.: Asynchronous games 2: The true concurrency of innocence. Theor. Comput. Sci. 358(2-3), 200–228 (2006)
5. Rideau, S., Winskel, G.: Concurrent strategies. In: LICS 2011. IEEE Computer Society (2011)
6. Clairambault, P., Gutierrez, J., Winskel, G.: The winning ways of concurrent games (submitted, 2012)
7. Winskel, G.: Events, causality and symmetry. Comput. J. 54(1), 42–57 (2011)
8. Abramsky, S., Melliès, P.A.: Concurrent games and full completeness. In: LICS 1999. IEEE Computer Society (1999)
9. Melliès, P.-A., Mimram, S.: Asynchronous Games: Innocence Without Alternation. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 395–411. Springer, Heidelberg (2007)
10. Abramsky, S.: Semantics of interaction. In: Pitts, A., Dybjer, P. (eds.) Semantics and Logics of Computation. Publications of the Newton Institute (1997)
11. Hyland, M.: Game semantics. In: Pitts, A., Dybjer, P. (eds.) Semantics and Logics of Computation. Publications of the Newton Institute (1997)
12. Abramsky, S., Jagadeesan, R.: Game semantics for access control. Electronic Notes in Theoretical Computer Science 249, 135–156 (2009)
13. Joyal, A.: Remarques sur la théorie des jeux à deux personnes. Gazette des sciences mathématiques du Québec 1(4) (1997)
14. Hintikka, J., Sandu, G.: A revolution in logic? Nordic J. of Phil. Logic 1 (1996)
15. Clairambault, P., Gutierrez, J., Winskel, G.: Imperfect information in logic and concurrent games (submitted, 2012)

Reflections on a \m/ Time with Dexter Kozen

Kamal Aboul-Hosn

Meridian America Inc.
110 Greene Street Suite 407
New York, NY 10012
kamal@sooloos.com

"Choose as an advisor someone you can see yourself being in ten years." Those were the words spoken to me by a fellow graduate student at Cornell University. It is a task that is easier said than done. How do I know who I want to be in ten years, let alone find someone who embodies those qualities? Through an incredible set of circumstances, I ended up working with an advisor who represented the person I wanted to be in ten years, Dexter Kozen. In the four and a half years I had the privilege of working with Dexter, I learned many lessons I have carried with me through graduate school and beyond to my life as a software engineer and manager.

My time with Dexter started with af teaching assistant position for his Theory of Computing class. Originally, I was a TA for the non-honors version of the class under a different professor. However, the honors version Dexter was teaching had a higher enrollment than expected, so they needed a TA to move from one class to the other. I was the one volunteered to move. I was not too excited about switching positions. As a first-semester graduate student who already felt out of his league at a university like Cornell, the idea of switching a couple of weeks into the semester to the honors version of a class was daunting. Nevertheless, TAing the course led to taking Dexter's Dynamic Logic course the next semester led to a research project in that class led to Dexter becoming my advisor.

The first quality that struck me about Dexter was his hands-off nature. In the Theory of Computing class, he'd leave coming up with a specific grading guide to the teaching assistants. As a graduate student doing work with him, it was often up to me to have the discipline to get the work done on research or papers that had deadlines. Dexter was never the kind of person to stand over my shoulder watching my every move. Instead, he would provide high-level guidance and expect me to carry the work out. Don't mistake the hands-off approach for a lack of interest; if I had questions at any time, he was always available. For a student trying to figure out how this whole research thing works, it can be overwhelming to have that much responsibility placed on you. However, the trial by fire encouraged me to develop the skills necessary to survive in a technical field, whether it be in academia or industry.

Seeing a hands-off approach from the other side when I became a manager gave me even more appreciation for how Dexter treated me as his graduate student. At the heart of this approach is trust. Dexter had to trust me to devote time to the projects and papers we were working on. He had to trust me to be thorough and rigorous in my work. And he had to trust me to seek his help

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 318–321, 2012.

when I had questions. That is a lot of trust to put in someone you barely know. It is the same trust I placed in each of my eight team members. When as an advisor or manager you place that trust in others from the beginning, you find out very quickly who are the people who are going to excel on their own and who are the ones who need more assistance. And you push those who have a natural inclination to excel to work even harder, for they understand the burden being placed on them. Dexter's trust brought out my best qualities and is the way I will always bring out the best qualities in those I work with.

As I mentioned earlier, it would be a mistake to think his hands-off approach meant Dexter was not interested. With its foundation in trust, the hands-off approach might even make one *more* interested; not performing up to the standards of a hands-off manager or advisor reflects a violation of the trust placed. I believe I only violated this trust on a couple of occasions, although you'll have to ask Dexter to confirm. I vividly remember one of the times my work did not match Dexter's standards. We were in the process of writing a paper to submit to the 2006 International Conference on Mathematics of Program Constructs (MPC). The paper required a lot of background information and definitions to set up the conclusions. However, I used a lot of terms that I did not define. I took what I thought was a nearly-complete paper to Dexter's office. He quickly found a lot of terms that were not defined and started marking them with his pen. As he found more, he got visibly frustrated, saying "Kamal, here's another one. You can't just use these terms without defining them." Dexter rarely let that kind of frustration show. When he did, I knew he was serious. Dexter taught me that a big part of being passionate about the work others are doing is getting frustrated occasionally and taking their errors personally.

Dexter's passion extended beyond his work and into his hobbies. I was fortunate enough to share in his musical interests while the drummer for the Joel Baines Trio, in which Dexter played guitar. I'm not sure there are many advisors who ask their students to join them in a rock band! Life outside of the office meant a relationship different from the typical advisor/advisee relationship for Dexter and me. We both had a love of music and overlapping music tastes, which is exactly what you want in a band. The vibe of band rehearsals and shows was much more relaxed than times in the office. That's not to say time in the office was stressful; it's just that we were both enjoying our passion for music, which had no deadlines, no proofs that needed to be completed, and no attempts to squeeze twenty pages of material into the fifteen-page limit for a conference paper. Instead, we spent time in a basement learning new songs that we all enjoyed, waiting for that moment when the song finally clicked and we could play it well. Often, those moments included a nod and a smile from Dexter as he strummed away on his guitar.

My favorite memories of my time in Ithaca are playing in the band with Dexter. In particular, I remember moments like playing the Computer Science Department Holiday Party in 2006. We had been asked to play an anniversary party for the department a few months earlier, but were abruptly stopped after two songs at the show because we were too loud. We were all unhappy about not

getting to play the full show we had planned. However, the holiday party was to be a much more appropriate venue for our band and its loud rock 'n roll. The rehearsal prior to the holiday party, we planned a little joke to start our show. We said that we had changed our music style in response to the anniversary party. After Dexter announced it, we started playing a slow doo-wop beat. The look of disappointment on the faces of my fellow grad students was obvious. After about 30 seconds of our new direction, we quickly broke into a wall of sound with "All The Small Things" by Blink-182. The dancing started and everyone had a great time. It remains for me one of the best shows I've ever been a part of.

Fig. 1. The Joel Baines Trio Perform at the Cornell Computer Science Department Holiday Party 2006. From left to right: Lee Armstrong (guest guitarist), Dexter Kozen (guitar, vocals), Joel Baines (bass, vocals), Kamal Aboul-Hosn (drums). Image courtesy of Stephanie Meik.

In sharing in music with Dexter, I learned another lesson from him: allow yourself the time to indulge in your hobbies. It has been my experience that the people who are passionate about their work are also passionate about their hobbies. For Dexter and me, it was making music. (And rugby for Dexter. That's one of those things I don't think my body could handle.) Dexter and I would often chat music before diving into a discussion about our latest proof. His taking an interest in me life allowed me to connect to him in a way that made working together more personal. I am thankful that Dexter not only took an interest in my favorite hobby, but that he had a similar passion we were able to share.

Vicky Weissman was right, "choose as an advisor someone you can see yourself being in ten years." Maybe that needs to be refined to say "choose as an advisor

someone you *want* to be in ten years." That doesn't necessarily mean choose an advisor whose career you want. It means choose an advisor who you respect as a human being. A person who exemplifies the qualities you one day hope to possess, both inside of the office and out. I am extremely fortunate and honored to have found the person I wanted to be in Dexter Kozen. It is obvious that he has taken the effort to be someone others could look up to. His lessons of trust, passion, and an enjoyment of life are ones we should all aspire to demonstrate as researchers in the field of Computer Science and as mentors to future Computer Scientists. Thank you, Dexter.

Two Three Pages Papers

Krzysztof R. Apt

CWI, Amsterdam and University of Amsterdam, The Netherlands

Dexter was my manager during the first 8 months or so of my one year stay at the IBM Research Center at Yorktown Heights in the mid eighties. Our interaction in terms of scientific output was very slim — it consisted of just a single paper, titled "Limits for automatic verification of finite-state concurrent systems." The paper appeared in 1986 in *Information Processing Letters* and had just 3 pages. Still, it has 275 citations on Google scholar, so more than 90 citations per page. I believe it is the best per page citation ratio for both of us.

I would like to mention here a short story that illustrates Dexter's style of being a manager. During my stay I invited for a visit my twin sister from Poland, which was then a communist country. To arrange it was not easy. One had to find a person with American nationality who would vouch for her financially, also in the case of medical problems. I summoned my courage, walked into Dexter's office and started to explain the problem. Before I reached the question whether *he* would be willing to vouch financially for my sister, he interrupted me and just asked: "where should I sign?". And that was all. I am forever grateful to Dexter for this, even though probably he completely forgot about it.

Perhaps what helped was that for Dexter Eastern Europe was not an empty concept but a place where some of his colleagues, like Andreka, Nemeti, and Tiuryn lived and worked. Thanks to Dexter my sister is now in possession of a historic photo I made her on the top of one of the Twin Towers.

During my stay at IBM Dexter decided to move to Cornell. Even though we did not maintain scientific contacts, I was watching with interest 'from the sidelines' Dexter's fine work on Kleene algebras. So I was particularly happy when he reacted positively to my invitation to contribute a paper to the new ACM Transactions on Computational Logic (ToCL) that I launched in 1999. Dexter's paper "On Hoare Logic and Kleene Algebra with Tests" appeared in the first issue. I may now reveal that I was one of the reviewers.

Strangely enough, the paper caused some problems for me. Namely, once I posted it on the website of ToCL somebody promptly contacted me claiming that he simplified a proof of one of Dexter's results. I suggested to publish his proof as a letter. However, in contrast to Dexter's crystal clear paper that I could read and understand without difficulties, the author wrote his note in such a cryptic and convoluted way, that I was totally lost. So I involved Dexter. In total 34 emails followed. (I still keep them.)

Fortunately for me, Dexter not only took the trouble to understand what the author wrote (which was really not obvious) but completely rewrote his arguments. This led to a short 3 pages paper of him and the author, that also appeared in the first issue of ToCL.

Dexter, many happy returns on your 60th birthday!

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, p. 322, 2012.

A Tribute from the Band

John Parker, Joel D. Baines, Paul Miller, and Julia Miller

Cornell University

Dexter has played music for most of his life and playing music is something that gives him great enjoyment. As a member of the band, Dexter has always been the organizer and musical director. Dexter is multitalented. He can play guitar, piano, and sing, all well enough to be done in public. As fellow band members we know when things are going well during a set because Dexter will have a huge grin on his face. Recently, Dexter has added Rapping to his repertoire. He has a really stellar performance that has yet to be presented to an audience!

Dexter is unabashedly and unequivocally exuberant for music. His kid like charm and enthusiasm has always been an inspiration to both his fellow band members and the audience. Dexter's enthusiasm for music makes it a blast to get together for practice and gigs. In addition to playing in the band, Dexter's enthusiasm stretches to acting as chief 'roadie'. Dexter trucks all of our equipment around in his van. This is a real effort as he has to load the van before the gig and then unload it when he gets home. For anyone who has ever wondered what that it is like - let me tell you it's a lot of work. Most guitar amps weigh at least 50-60 lb so loading and unloading is hard manual labor. From all of us in the band, Happy 60th birthday Dexter. I think that I speak for all the members of the current band and past bands in saying that it is a total blast to play music with you! Here's a toast to you and to playing music with you for many more years.

All the best. *The Band.*

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, p. 323, 2012.

Dexter Kozen: An Appreciation

Joseph Y. Halpern

Computer Science Department
Cornell University
Ithaca, NY 14853, USA
halpern@cs.cornell.edu

I have known Dexter for 30 years, which, of course, means that I was 9 when I first met him. At that time, Dexter was already "the man". Although he was only a few years ahead of me, he had already made his name by independently defining the notion of alternating Turing machines, a deep contribution to complexity theory that made it possible to connect time and space complexity. The results were viewed as so significant that it was already being taught in a graduate course on complexity theory that I attended.[1] Of even more interest to me at the time was Dexter's work on modal logic, since a large part of my thesis was on dynamic logic. Finding a sound and complete axiomatization for dynamic logic had been an open problem for many years. Krister Segerberg had suggested an obviously sound axiomatization, but couldn't prove it complete. Rohit Parikh [7] showed that it was indeed complete, but his proof was rather complicated. Dexter then came up with a much simpler proof, one that got at the essence of Rohit's ideas; this version was published as [6]. The proof is truly beautiful. I have taught it often, and used the ideas in a number of subsequent papers (e.g., [3,4]).

This was my first introduction to Dexter's ability to see into the heart of a problem. Perhaps the best-known example is Dexter's work on the μ-calculus. Vaughan Pratt [8] had earlier suggested a logic with fixed-point operators. Dexter's version [5] was more elegant, and is the one that everyone focuses on these days.

I also soon learned about Dexter's breadth. Besides complexity theory and modal logic, he also produced major results on algebra, such as the complexity of the theory of real closed algebraic theories [1]. I remember visiting IBM Yorktown Heights in the mid 1980s; Dexter was working there at the time. I mentioned to him some graph-theoretic problems I was working on. The rest of the day, I could see Dexter scratching away at a notepad, thinking about the problems. (He is still scratching away at notepads, although the notepad and the problems occupying him are no doubt different.)

When I moved to Cornell in 1996, I saw a different side of Dexter. Dexter is a true department stalwart. He is well known to be a tremendous teacher. He can also always be counted on to play hockey or play music at a department function. (On top of everything else, Dexter is a terrific musician—I am

[1] Ashok Chandra and Larry Stockmeyer had also investigated alternation; they joined forces to produce a very high-cited journal paper [2].

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 324–325, 2012.

jealous!) Now that I'm department chair, I appreciate Dexter's contributions to department life even more. Each year I ask department members what courses they would like and be willing to teach. Most people mention two courses; some people may even mention three. Dexter is willing and able to teach just about anything: theory courses, programming language courses, all of our introductory programming and data structures courses, the basic graduate courses in theory and programming languages, and, of course, specialty courses in his own area. Indeed, not only is he willing to teach them in principle, he has in fact taught just about all of them.

When one of our faculty members had a sudden emergency the night before the final exam in his core undergraduate course (a course with over 60 students), with the final exam only partially completed (yes, it was the night before the final . . .), he called on Dexter. Dexter stayed up to the wee hours of the morning preparing the final, and helped grade it. I think this incident really gives a sense of Dexter—both the way he came through for someone else in an emergency, and that he was the one that was turned to in the first place.

Thanks for everything, Dexter.

References

1. Ben-Or, M., Kozen, D., Reif, J.H.: The complexity of elementary algebra and geometry. Journal of Computer and System Sciences 32(1), 251–264 (1986)
2. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. Journal of the ACM 28, 114–133 (1981)
3. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences 30(1), 1–24 (1985)
4. Halpern, J.Y., Moses, Y.: A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence 54, 319–379 (1992)
5. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27(1), 333–354 (1983)
6. Kozen, D., Parikh, R.: An elementary proof of the completeness of PDL. Theoretical Computer Science 14(1), 113–118 (1981)
7. Parikh, R.: The Completeness of Propositional Dynamic Logic. In: Winkowski, J. (ed.) MFCS 1978. LNCS, vol. 64, pp. 403–415. Springer, Heidelberg (1978)
8. Pratt, V.: A decidable μ-calculus. In: Proc. 22nd IEEE Symposium on Foundations of Computer Science, pp. 375–383 (1981)

Dexter Kozen: A Winning Combination of Brilliance, Depth, and Elegance

David Harel

This is a somewhat non-standard piece about Dexter Kozen, longtime colleague and friend. I am sure that the praise of Dexter's research will have been sung by many of the people contributing to this volume. I could probably have added my own perspective, talking at length about the enormous impact of his most profound work. For example, the wonderful paper on alternation influenced my work on seemingly unrelated topics years later, in many unexpected ways.[1]

However, on second thoughts, I decided to do it a little differently. I'm going to tell you why I decided not to write a paper for this volume in honor of Dexter, but instead to write a *laudatio*. A paper would have been easy, right? We all do research and write papers, and it shouldn't be too much of a problem to select a fitting one for the occasion. So, why not?

Well, there are two reasons: The first is that any paper I could have written, which would have been in some way relevant to a volume in honor of Dexter Kozen, would have fallen short, in the following sense: Dexter would have read it and would probably have thought: "Oh yea, that's kind of interesting, and yes, I can see how those proofs go; and, by the way, here is how to continue the work and get far stronger results, and in a much nicer way". (This is the best case, of course; it could be a lot worse. Dexter sees errors, gaps in proofs, shaky arguments, etc., very quickly…)

The second reason is a lot more acute. I simply can't write a paper befitting this volume, because I stopped doing theory of computation almost twenty years ago. Why? Well, that's an easy one: It's Dexter. He's the one to blame.

Let me explain: Dexter Kozen is absolutely amazing! IMHO, he is one of the most brilliant theoretical computer scientists of our generation. But it's not just his cleverness and depth and the wonderful work he's been able to produce, but also the unparalleled beauty of his thinking and of the way he goes about doing his research. A work session with Dexter was always a combination of exhilaration and frustration. He had the uncanny ability to bring in, out of the blue, unexpected notions from seemingly unrelated branches of mathematics – often from algebra or topology – which either you'd have never heard of, or you'd have long-forgotten. And these then turned out not only to be relevant but to make everything you wanted to do fall into place, and in a concise, exotically elegant and often stunning way.[2] Such collaboration

[1] As is well-known, Dexter's work on alternation appeared initially in his singly-authored *FOCS`76* paper, independent of the Chandra-Stockmeyer paper that was published back-to-back with it in the same volume; the two later became the famous combined, triply-authored *J.ACM* version.

[2] For me, an excellent example of this was his introduction of ultra-filters into our work on dynamic logic.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 326–327, 2012.

was, of course, exciting and extremely fruitful, but could also be frustratingly depressing. By the way, Dexter would later go off on beautiful tangents, open up new avenues, ask new questions and then obtain more, and stronger, results. At that point most of us simply give up…

As a result, I figured that someone who produced the most succinct and beautiful proof imaginable of completeness for PDL[3], who went on to provide a virtuosic treatment of the far more challenging μ-calculus, who placed logics of programs in an elegant Kleene-like algebraic setting, and on and on and on (and all this without mentioning his fundamental contributions to complexity theory, and a whole slew of more recent work that I haven't been able to follow); such a person causes one to want to become a taxi driver….

No way could I do theory that would even come *close* to what Dexter was able to produce in his seemingly effortless way, out of his sleeve. So I quit. Not to become a taxi driver, but to do different things, which require far less of the qualities that Dexter had in such amazing abundance.

*

What else can I say? Please accept my heartfelt wishes, Dexter, for many more fruitful years of scientific productivity, and health, joy and bliss. Enjoy your family and enjoy life!

I am proud to know you and to have been able to work with you and to learn from you.

[3] Dexter's work was based on ideas from Rohit Parikh's original proof of the completeness of the Segerberg axioms for PDL; the two ended up publishing jointly.

Making the World a Better Place

John Hopcroft

Cornell University

The papers in this book detail Dexters research contributions, which are profound. I would like to highlight another contribution Dexter has made that affects us all in the department every day.

Dexter has a deep concern for the well being of our department. He can always be counted on to take the steps necessary to create a smooth running arena where students can flourish. He cares about their success and makes himself available to help them in any way he can. If something needs doing, he volunteers and gets the job done. He's a master at cutting red tape. He will teach whatever the department needs. Dexter goes for quality in everything he does. When he evaluates a person, he reads the persons paper and asks what the individual has done, not how many papers were published.

I feel that, overtime, Dexter's values have become absorbed by the department and are reflected in quality work, respect for others and a sincere collegiality. He is the professor you hope for, whether as a student or a colleague.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, p. 328, 2012.

Timesharing Dexter

Susan Landau*

My husband Neil Immerman returned from the 1986 STOC meeting with an interesting proposition. Juris Hartmanis and Dexter Kozen had a small pocket of funds, and they proposed that the two of us visit the Cornell Computer Science Department for a week.

It sounded delightful but we had a complication: our new son, who was all of four months old. We decided to time-share Dexter and split child care (using the rule that during the day, he who was not with the baby would be with Dexter). The Kozens improved upon this, offering that we could stay at their house. So while one of us would be talking research with Dexter in his office, the other would be taking care of the baby while visiting Fran at home.

Thus began Dexter's and my adventure into polynomial decomposition. The week before I arrived at Cornell, I had been thinking about polynomial decomposition, that is, the issue of finding a non-trivial solution to the problem $f(x) = g(h(x))$ (non-trivial means that both $g(x)$ and $h(x)$ are of degree greater than 1). Barton and Zippel had a solution for fields of characteristic 0, noting that if $f(x) = g(h(x))$, then $h(x) - h(y)$ divides $f(x) - f(y)$. They used this — and a refinement, under the assumption that $h(0) = 0, h(x)|(f(x) - f(0))$ (without loss of generality, one can assume that $h(0) = 0$) — to find potential $h(x)$ [2]. Their algorithm was exponential in n, the degree of $f(x)$.

Even with the lack of sleep that accompanies having a baby, I thought I could do better. Lüroth's theorem states that if k is an arbitrary field, the fields between $k(f(x))$ and $k(x)$ are in one-to-one correspondence with the decompositions of $f(x)$. Each field between $k(f(x))$ and $k(x)$ can be expressed as $k(h(x))$ for some composition factor of $f(x)$ [7].

I knew how to find certain subfields of a field rather quickly [6] and I thought I could apply that technique. But my potential solution ran into a difficulty. Instead of being kept awake by our son, I spent my first night in Ithaca awake puzzling over polynomial decomposition and blocks of roots of polynomials. That Monday afternoon I talked with Dexter about the problem, my approach, and the difficulty with it. Dexter hadn't been thinking about polynomials, decomposition, or subfields, but in his inimitable fashion, Dexter immediately got very excited. We got to work.

Let me provide some notation and background. Let k be a field of arbitrary characteristic and let $f(x)$ be a monic separable polynomial (no repeated roots) of degree n with coefficients in k. Let K be the splitting field of $f(x)$ over k, the smallest field containing all the roots of $f(x)$ over k. Futhermore let G be the Galois group of $f(x)$ over k, the set of permutations of the roots that hold the base field k fixed.

* Visiting Scholar, Department of Computer Science, Harvard University.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 329–332, 2012.

Evariste Galois showed that there is a one-to-one correspondence between the subgroups of G and the subfields between K and k. (He used this to show that roots of arbitrary polynomials of degree five or greater are not necessarily expressible in radicals.) From previous work [6], as long as $f(x)$ was irreducible over a field of characteristic 0 (and $k[x]$ had a factoring algorithm), I had an efficient method for for computing the fields that lay between k and $k[x]/f(x)$.

My work relied on *block decomposition*. If G is a permutation group on $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, the roots of $f(x)$ over k, we let G_α be the subgroup of G that fixes α. The fields between k and $k[x]/f(x)$ — one of which was $k[x]/h(x)$ — correspond to subgroups of G_α. Finding intermediate fields could give a decomposition. But decomposable polynomials may have repeated roots, and Galois fields don't capture this situation.

I thought our week in Ithaca would involve Neil in Dexter's office in the mornings, me there in the afternoons, while the evenings would have Fran, Dexter, Neil and me at home, visiting. I had that partially right. Fran, Dexter, Neil and I were at home in the evenings, and sometimes we all got to visit (mostly over dinner). But on decomposition Dexter was like a dog with a bone: toss the repeated roots problem in the air, let it land, grab it, worry it some more, toss it again, and keep it going. He and I spent the evenings puzzling over, pulling at, pressing on decomposition.

If the approach of Galois fields wouldn't allow repeated roots, generalizing the notion of blocks would.

Let k be a field of arbitrary characteristic and let $f(x)$ be a monic polynomial in $k[x]$ of degree $n = rs$, with $f(x)$ not necessarily irreducible or separable. Let K be the splitting field of $f(x)$ over k, and let G denote the Gaiois group of $f(x)$ over k. Dexter and I defined a *block decomposition* for $f(x)$ a multiset A of multisets of elements of k such that,

- $f(x) = \Pi_{A \in \Delta} \Pi_{\alpha \in A}(x - \alpha)$;
- if $\alpha \in A \in \Delta$ and $\beta \in B \in \Delta$, and $\sigma \in G$ is such that $B = \sigma(A) = \{\sigma(\rho) | \rho \in A\}$.

A block decomposition Δ is an $r \times s$ block decomposition if $|\Delta| = r$ and $|A| = s$ for all $A \in \Delta$. This generalization of block decomposition to multisets meant that $f(x)$ could have repeated roots. Dexter was very happy (the bone stopped being tossed in the air quite so often). This definition enabled Dexter and me to generalize the subfield issue to handle reducible polynomials and polynomials with repeated zeros. Before I present our structure theorem, I need to provide some additional notation for you to gnaw on:

Let:

$f(x) = x^n + a_{rs-1}x^{rs-1} + \ldots + a_0$, with $a_i, 0 \leq i \leq rs - 1$;
$g(x) = x^r + b_{r-1}x^{r-1} + \ldots + b_0$, with $b_j, 0 \leq j \leq r - 1$; and
$h(x) = x^s + x_{s-1}x^{s-1} + \ldots + c_0$, and with $c_k, 0 \leq k \leq s, \in k$.

Furthermore let c_j^m denote the jth elementary symmetric function on m-element multisets:

– $c_j^m = \Sigma_{B \subseteq A, |B|=j} \Pi B$, and
– $c_m = 1$.

Dexter and I showed:

Theorem: Let $f(x) \in k[x]$ be monic of degree $n = rs$. The following two statements are equivalent:

– $f(x) = g(h(x))$ for some $g(x)$ and $h(x)$ in $k[x]$ of degree r and s respectively.
– There is an $r \times s$ block decomposition Δ for $f(x)$ such

that $c_s^j(A) = c_s^j(B) \in k$ for all $A, B \in \Delta,\ O \leq j <\leq s-1$ [5].

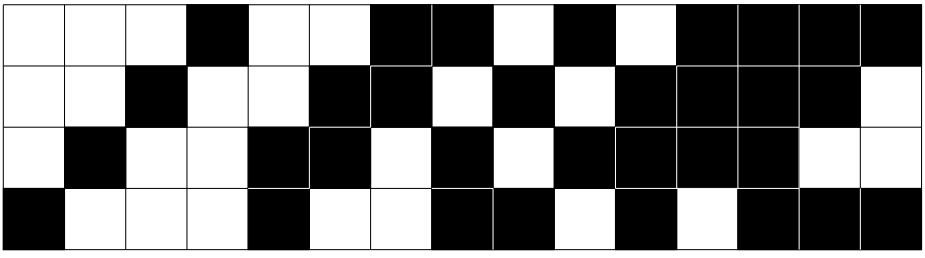
Without loss of generality we can assume that $c_0 = 0$. With that assumption, we get that if $f(x) = g(h(x))$, then $f(x)$ and $h(x)$ agree on their first r coefficients. The calculation of the remaining coeffcients of $h(x)$ falls out from the simple recurrence equation for the c_i. From $h(x)$ we can determine $g(x)$. (Because the system is overdetermined, we have to check that candidate $g(x)$ and $h(x)$ actually lead to a decomposition.) Our algorithm decomposes $f(x)$ in $O(n^3)$ in general — a rather impressive improvement from the earlier exponential-time algorithms. The algorithm works even faster if the underlying field supports Fast Fourier Transform ($O(n^2 \log n)$ steps) [5].

The bone had been fully gnawed upon. Dexter was delighted (as was I). There was more to come.

My motivation in considering decomposition was because of its fundamental role in computer algebra. But Dexter's and my result turned out to have other applications as well. In 1985 a cryptosystem was proposed based on polynomials [3]. Because in composition polynomial degrees multiply (rather than add, as is the case for polynomial multiplication), the thought was that perhaps composition could be an RSA-type cryptosystem based on polynomials.

The Kozen-Landau theorem shows that polynomial composition is not a good candidate for such public-key systems. Recently I was told that in the main Maple command "solve" for solving polynomial systems (and pretty much everything else), the algorithm begins by attempting to decompose any polynomials passed as input. This is because even while few polynomials are decomposable, the decomposition method is sufficiently fast that it provides a big win when it succeeds. The implementation is the Kozen-Landau technique [4].

There was yet another consequence of Kozen-Landau. The polynomial $x^4 + x + 1$ is the smallest polynomial over $GF(2)$ that has a non-trivial decomposition: $x^4 + x + 1 = g(h(x))$, with $g(x) = x^2 + x + 1$ and $h(x) = x^2 + x$. Sometime after Neil and I left Ithaca and the Kozen domicile, Fran and Dexter retiled their bathroom shower. They included a strip of 4×4 small green and white tiles running along the wall; it is the cyclic multiplicative group of $GF(16)$ as represented by polynomials $\bmod(x^4 + x + 1)$ generated by x:



Even when he showers, Dexter can't get away from decomposing polynomials!

References

1. Alagar, V.S., Thanh, M.: Fast Polynomial Decomposition Algorithms. In: Caviness, B.F. (ed.) EUROCAL 1985, Part II. LNCS, vol. 204, pp. 150–153. Springer, Heidelberg (1985)
2. Baron, D., Zippel, R.: Polynomial Decomposition Algorithms. Journal of Symbolic Computation 1, 159–168 (1985)
3. Cade, J.J.: A New Public-Key Cipher Which Allows Signatures. In: Proceedings of Second SIAM Conference on Applied Linear Algebra, Raleigh, NC (1985)
4. Giesbrecht, M.: Personal Communication, December 1 (2010)
5. Kozen, D., Landau, S.: Polynomial Decomposition Algorithms. Journal of Symbolic Computation 7, 445–456 (1989)
6. Landau, S., Miller, G.L.: Solvability by Radicals is in Polynomial Time. Journal of Computer Systems Science 30, 179–208 (1985)
7. van der Waerden, B.L.: Algebra. Frederick Ungar Publishing Co. (1977)

A Small Tribute

Anil Nerode

Cornell University

I have known and admired Dexter since the time long ago when Juris Hartmanis was chairman of his PhD committee and I was a member. Shortly after he came back to Cornell as a professor an occasion arose when he thought to ask why I had not attended his qualifying exam. I told him that Juris had to because he was chair, but that I avoided exams that were superfluous. I am not much on "rites of passage".

The originality, breadth, and depth of Dexters subsequent research in such areas as Kleene algebras, automata theory, dynamic logic, and the mu calculus has never ceased to amaze me.

Our main personal contact over the years has been through our common membership on the three person PhD committees which guide our graduate students to their degrees. There has been for many years an informal logic-related group in mathematics and computer science. It has consisted of Richard Shore and myself in Mathematics and Dexter Kozen and Bob Constable in computer science, recently joined by Joseph Halpern. We have served as members of many PhD committees in mathematics and in computer science, to the benefit of both fields.

Dexter has been a principal contributor to a distinguished several decade history of educating mathematical logicians with strong research credentials in computer science and computer scientists with strong research credentials in mathematics. May he have a long and productive future!

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, p. 333, 2012.

Dexter Kozen's Influence on the Theory of Labelled Markov Processes

Prakash Panangaden

School of Computer Science, McGill University

In the Fall of 1985 Dexter and I both started at Cornell as new faculty members in the celebrated Computer Science Department, home to luminaries such as Juris Hartmanis, John Hopcroft, David Gries and Robert Constable. I was a very new assistant professor but Dexter was already an acknowledged star with celebrated contributions to several areas: algebra and complexity, decision procedures for real-closed fields [1], dynamic logic [2–4] and many other areas across both tracks of theoretical computer science. I had no doctorate in computer science, hardly any publications and no clearly defined research area. Early in the term Dexter summoned me to his office and grilled me about work I was doing on nondeterministic dataflow. After that meeting I needed several glasses of beer to recover but a lasting friendship was sealed.

One of the first things Dexter did was to offer a series of informal lectures on his work on the semantics of probabilistic programs and on probabilistic PDL [5, 6]. I attended these lectures and was dazzled and bewildered. This was not the probability theory I had seen being used in computer science in those days; or for many years after! He plunged right into measure theory, σ-algebras, Stone-type dualities and other things that I had never heard of. I struggled to follow but with the pressing need to define my own research program I concentrated more and more on concurrency theory, type theory, epistemic logic and denotational semantics and could not spend time thinking about all the things Dexter was explaining. It was a full decade later that those lectures had their impact by which time I had been away from Cornell for several years.

Let me explain two key ideas that I learned subconsciously which influenced me later. First, there is a striking Stone-type duality [7] between two kinds of programming language semantics: the ordinary state transformation semantics one sees in operational semantics or denotational semantics and the backward flowing predicate-transformer semantics due to Dijkstra [8]. This duality was discovered by Plotkin [9] and Smyth [10]. Dexter, influenced by suggestions made by Plotkin, discovered an analogous duality in probabilistic semantics.

In order to sketch the ideas I will define the language used in the original paper. We define the syntax as follows:

$$S ::== x_i := f(\boldsymbol{x}) | S_1; S_2 | \mathit{if}\ \mathbf{B}\ \mathit{then}\ S_1\ \mathit{else}\ S_2 | \mathit{while}\ \mathbf{B}\ \mathit{do}\ S.$$

We assume that the program has a fixed set of variables $\boldsymbol{x}$, say n distinct variables, and that they each take values in some measure space (X, Σ). In his first paper [5], Dexter gives a state transformer semantics by assigning commands to Markov kernels. Crucially, one can compose Markov kernels by integrations and

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 334–337, 2012.

one needs this for a compositional denotational semantics. The dual view is to think of random variables being transformed backwards through the commands. If the value of a random variable is given at some program point what is the value before the preceding command? This is just the probabilistic analogue of the predicate transformer point of view.

This point of view was explored extensively by the Oxford group in the early 1990s, see for example the recent book by Morgan and McIver [11].

Many years later I, working with Abramsky and Blute, was seeking quantitative generalizations of the notion of relational composition and was seeking a category that like **Rel** was compact closed but worked by combining functions by integration instead of combining predicates by existential quantification. One composes two binary predicates $R(x, y)$ and $S(x, y)$ by the simple formula

$$(R \circ S)(x, y) = \exists z\ (R(x, z) \wedge R(z, y)).$$

It seemed natural to think of a probabilistic relation as a joint distribution on the product space $X \times Y$ but how does one compose them?

Eventually, I realized that the right gadgets were Markov kernels and I was able to define a category of probabilistic relations [12]. If only I had remembered Dexter's lectures I would have saved months of time. I called this category **SRel** and shortly afterwards discovered that Gìry had developed a theory of probability monads much earlier [13] and **SRel** is the Kleisli category of this monad.

It still did not yield a category with all the properties of relations, but we eventually realized that this is a feature not a bug, and we developed the theory of nuclear ideals [14]. Once the category **SRel** was in place I was able to give a simplified treatment of Dexter's semantics and his duality, which I called Kozen duality [15].

The most striking idea that I learned from Dexter is the analogy between logic and probability: in fact the duality mentioned above is just an echo of this underlying idea. This idea is best conveyed through the following table.

Classical logic	Generalization
Truth values $\{0, 1\}$	Probabilities $[0, 1]$
Predicate	Random variable
State	Distribution
The satisfaction relation $\models$	Integration $\int$

Just as the satisfaction relation, $\models$, links states and formulas to give truth values, so the integral links distributions (generalized states) with random variables (generalized formulas) to give numerical results (generalized truth values).

Around 1994 I began a long series of investigations into the theory of what I called labelled Markov processes [16–26]. One of the contributions of this work was the development of a metric analogue of bisimulation. We defined a metric, or more precisely a pseudometric with probabilistic bisimulation as its kernel. How could one do this? The construction is based on an idea that comes straight

from Dexter's analogy above. We defined a real-valued logic which, like an ordinary modal logic characterized bisimulation in the spirit of van Benthem and Hennessy-Milner. Now we can define a distance between states s, t of a probabilistic transition system by

$$d(s,t) = \sup_f |f(s) - f(t)|$$

where the sup is over all the formulas of the logic. Dexter's ideas have been incredibly fruitful and we are still mining them. Meanwhile, though Dexter has been busy with many other topics he came up with a beautiful coinduction principle for stochastic processes in a recent [27, 28] paper. Perhaps by 2017 I will understand it well enough to develop the subject further. Meanwhile, happy birthday Dexter!

Acknowledgments. My work on LMPs has been greatly stimulated by my collaborators Samson Abramsky, Rick Blute, Philippe Chaput, Abbas Edalat, Norm Ferns, Vineet Gupta, Chris Hundt, François Laviolette, Gordon Plotkin, Doina Precup and most especially Vincent Danos, Josée Desharnais and Radha Jagadeesan.

References

1. Ben-Or, M., Kozen, D., Reif, J.: The complexity of elementary algebra and geometry. Journal of Computer and System Sciences 32, 251–264 (1986)
2. Harel, D., Kozen, D., Tiuryn, J.: Propositional Dynamic Logic. MIT Press (2000)
3. Kozen, D., Parikh, R.: An elementary proof of the completeness of PDL. Theoretical Computer Science 14, 113–118 (1981)
4. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983)
5. Kozen, D.: Semantics of probabilistic programs. Journal of Computer and Systems Sciences 22, 328–350 (1981)
6. Kozen, D.: A probabilistic PDL. Journal of Computer and Systems Sciences 30, 162–178 (1985)
7. Johnstone, P.: Stone Spaces. Cambridge Studies in Advanced Mathematics, vol. 3. Cambridge University Press (1982)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
9. Plotkin, G.D.: Lecture notes on domain theory. The Pisa Notes (1983), `http://www.dcs.ed.ac.uk/home/gdp/publications/pubs.html`
10. Smyth, M.: Powerdomains and Predicate Transformers. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 662–676. Springer, Heidelberg (1983)
11. Morgan, C., McIver, A.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer, Heidelberg (2004) (to appear)
12. Panangaden, P.: The category of Markov processes. ENTCS 22, 17 pages (1999), `http://www.elsevier.nl/locate/entcs/volume22.html`
13. Giry, M.: A categorical approach to probability theory. In: Banaschewski, B. (ed.) Categorical Aspects of Topology and Analysis. Lecture Notes In Mathematics, vol. 915, pp. 68–85. Springer, Heidelberg (1981)

14. Abramsky, S., Blute, R., Panangaden, P.: Nuclear and trace ideals in tensor-$*$ categories. Journal of Pure and Applied Algebra 143, 3–47 (1999)
15. Panangaden, P.: Labelled Markov Processes. Imperial College Press (2009)
16. Blute, R., Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labelled Markov processes. In: Proceedings of the Twelfth IEEE Symposium On Logic In Computer Science, Warsaw, Poland (1997)
17. Desharnais, J., Edalat, A., Panangaden, P.: A logical characterization of bisimulation for labelled Markov processes. In: Proceedings of the 13th IEEE Symposium on Logic in Computer Science, pp. 478–489. IEEE Press, Indianapolis (1998)
18. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for Labeled Markov Systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 258–273. Springer, Heidelberg (1999)
19. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximation of labeled Markov processes. In: Proceedings of the Fifteenth Annual IEEE Symposium on Logic in Computer Science, pp. 95–106. IEEE Computer Society Press (2000)
20. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labeled Markov processes. Information and Computation 179, 163–193 (2002)
21. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: The metric analogue of weak bisimulation for labelled Markov processes. In: Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science, pp. 413–422 (2002)
22. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Weak Bisimulation is Sound and Complete for PCTL*. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 355–370. Springer, Heidelberg (2002)
23. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximating labeled Markov processes. Information and Computation 184, 160–200 (2003)
24. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: A metric for labelled Markov processes. Theoretical Computer Science 318, 323–354 (2004)
25. Danos, V., Desharnais, J., Laviolette, F., Panangaden, P.: Bisimulation and cocongruence for probabilistic systems. Information and Computation 204, 503–523 (2006)
26. Chaput, P., Danos, V., Panangaden, P., Plotkin, G.: Approximating Markov Processes by Averaging. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 127–138. Springer, Heidelberg (2009)
27. Kozen, D.: Coinductive proof principles for stochastic processes. In: Alur, R. (ed.) Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, LICS 2006, pp. 359–366 (2006)
28. Kozen, D.: Coinductive proof principles for stochastic processes. Logical Methods in Computer Science 3, 1–14 (2007)

An Appreciation of Dexter Kozen

Rohit Parikh

March 7, 2012

I met Dexter Kozen in October 1979 at the FOCS meeting in Puerto Rico where he gave a paper on the semantics of probabilistic programs. The main reason for our discussion at that meeting was the fact that he was the referee of a paper which I had submitted to *Theoretical Computer Science - TCS* with Albert Meyer as editor.

Here is the background, some of which overlaps what Joe Halpern has said. After Pratt's proposal [13] and development of (first order) Dynamic Logic as a way of providing a semantics for programs and proving their correctness, Fischer and Ladner approached the related issue of *propositional* Dynamic Logic (PDL) [2]. They proved decidability, and exponential time completeness[1] (using a technique due to Chandra, Kozen and Stockmeyer [1] for the lower bound). But they left the problem of completeness open. This was in the late 70's.

I was then a professor at Boston University but fairly close to the group at the Laboratory of Computer Science at MIT, especially with my friends Albert Meyer, Vaughan Pratt and David Harel. Influenced by the trio I started to work on the completeness of PDL and published my solution in the proceedings of a conference in Zakopane, Poland [9].[2]

I then submitted the paper to Albert Meyer for publication in *TCS* and Meyer astutely chose Kozen as the referee. This was the background for my chats with Dexter in Puerto Rico and later at IBM. This incident itself illustrates a trait of Dexter, that he is so straightforward. Referees in general are eager to hide their identity, but Dexter had no trouble communicating openly with me.

Try as he could, Dexter could not follow my proof, and I confess that the fault was not entirely his. The diffculty in proving completeness was this. There were standard techniques for proving completeness of modal logics via the construction of a canonical model. The issue was with the Kleene[3] star operator. The usual construction interpreted the *-operator in a possibly non-standard way so that the *-closure of a relation R might involve infinitely many iterations (past ω). This meant that completeness was obtainable for *nonstandard* models of PDL, but how to show that PDL was also complete relative to *standard* models? (Note: completeness of a formal system relative to a larger class of models does not imply completeness relative to a smaller class.) Dexter did not enjoy my elaborate constructions to convert a nonstandard model into a standard one,

[1] [2] only proved a non-deterministic exponential upper bound. A deterministic exponential upper bound was proved by Pratt [14].

[2] Krister Segerberg announced a set of axioms while I was working on completeness, but it turned out that he did not actually have a completeness proof for his axioms at that time.

[3] Kleene himself was present at that meeting in Puerto Rico.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 338–340, 2012.

and eventually came up with a brilliant solution. He bypassed most of my construction and found an argument using the same ideas, but directly constructing a finite standard model. Thus we had the Kozen-Parikh completeness proof of PDL [4], widely cited and read in the literature.

My collaborations with Dexter continued through several papers, [10, 5, 6] and our friendship also became more personal when I moved to CUNY, living first in Brooklyn and then in Larchmont. I still remember an occasion when Dexter was visiting us in Larchmont and we wanted to go to the Larchmont Manor park on Long Island Sound. But there were five of us and only room for four in the car. Dexter, not a small man, gamely offered to ride in the trunk and did! (The car was a hatchback so the trunk was not quite a prison).

My collaborations with Dexter ceased after 1983, but I did not lose interest in our common area. Some time in the early and middle 80's I realized that PDL could be converted into a logic to reason about games by simply dropping one of the axioms (the PDL equivalent of Kripke's normality axiom). I adapted the Kozen-Parikh completeness proof to the logic of games and that paper [11] has also been influential.

However, the completeness I proved was for the "dual-free" part of the logic of games and as far as I know, the completeness of the axioms I proposed then for the *full* game logic has been neither proved nor disproved.

From the logic of games I proceeded to work in epistemic logic, and in game theory proper, trying to develop a logician's understanding of how games and society work. A good early reference is [12]. But that is a part of my career which no longer involves Dexter.

Perhaps I can say a little bit about this area. The work I did with Dexter and others was essentially about the mathematics of programs. But society itself consists of lots of programs. There are single-agent programs like a cooking recipe or learning a language or travelling to a conference. In such programs, either there are no other agents (the first two cases) or if there are other agents, they are assumed co-operative and one can treat them as resources no different from subroutines.

In other social programs which are inherently multi-agent, like elections or wars, it is assumed not only that there are other agents, but that their goals may be incompatible or only partially compatible. And in that case, anticipating their moves is an additional issue.

A deep understanding of mathematical programs was developed by the dynamic logic community (consisting of Dexter and I and many others) and the temporal logic community. But some of these insights can apply also to the "software" of society. So I hope that some of the brilliant minds who have worked on the logic of computer programs will turn their attention to the logic of social programs. A good survey can be found in [7]. See also [16]

Finally, I should mention a professional contact between Dexter and me which was not a personal contact. Some time in the 90's Dexter became interested in Parikh's theorem, a result which I had proved for Chomsky when I was a graduate student at Harvard. See [3, 8]. But this research was done entirely at Dexter's

own initiative. I myself had become more interested in social and philosophical issues, and Dexter the algebraist was much more of a mathematician than I was at that stage.

References

[1] Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. Journal of the ACM (JACM) 28(1) (January 1981)
[2] Fischer, M., Ladner, R.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18, 194–211 (1979)
[3] Hopkins, M., Kozen, D.: Parikh's Theorem in Commutative Kleene Algebra. In: Proc. 14th Conf. Logic in Computer Science (LICS 1999), pp. 394–401. IEEE (July 1999)
[4] Kozen, D., Parikh, R.: An Elementary Completeness Proof for PDL. Theor. Comp. Sci. 14, 113–118 (1981)
[5] Harel, D., Kozen, D., Parikh, R.: Process Logic: Expressiveness, Decidability, Completeness. JCSS 25, 144–170 (1982)
[6] Kozen, D., Parikh, R.: A Decision Procedure for the Propositional μ-calculus. In: Clarke, E., Kozen, D. (eds.) Logic of Programs 1983. LNCS, vol. 164, pp. 313–325. Springer, Heidelberg (1984)
[7] Pacuit, E., Parikh, R.: Social Interaction, Knowledge, and Social Software. In: Goldin, D., Smolka, S., Wegner, P. (eds.) Interactive Computation: The New Paradigm, pp. 441–462. Springer, Heidelberg (2007)
[8] Parikh, R.: Language generating devices, MIT Research Laboratory for Electronics, Quarterly Progress Report, pp. 199–212 (January 1961) (republished as On Context Free Languages. Jour. ACM 13, 570-581 (1966))
[9] Parikh, R.: The Completeness of Propositional Dynamic Logic. In: Winkowski, J. (ed.) MFCS 1978. LNCS, vol. 64, pp. 403–415. Springer, Heidelberg (1978)
[10] Parikh, R.: Propositional Logics of Programs: A Survey. In: Engeler, E. (ed.) Logic of Programs 1979. LNCS, vol. 125, pp. 102–144. Springer, Heidelberg (1981) (one section of thie paper was written by Dexter, but he refused to be co-author, contenting himself with an acknowledgement within the paper)
[11] Parikh, R.: The Logic of Games and its Applications. Annals of Discrete Math. 24, 111–140 (1985)
[12] Parikh, R.: Social Software. Synthese 132, 187–211 (2002)
[13] Pratt, V.: Semantical considerations on Floyd-Hoare Logic. In: Proceedings of the 17th Symposium on Foundations of Computer Science, pp. 109–121 (1976)
[14] Pratt, V.: A practcal decision method for propositional dynamic logic. In: Proceedings of 10th ACM Symposium on Theory of Computation, pp. 326–337 (1978)
[15] Segerberg, K.: A completeness proof for the modal logic of programs (preliminary report). Notices of the American Mathematical Society 24, A-552 (1977)
[16] Wikipedia entry on Social Software, `http://en.wikipedia.org/wiki/Social_software_social_procedure`

To Dexter - A Tribute from Aarhus

Erik Meineche Schmidt, Mogens Nielsen, and Sven Skyum

Department of Computer Science, Aarhus University, Denmark

At Aarhus University, we consider Dexter to be one of our oldest and best friends and colleagues. Dexter has visited our Department of Computer Science as a guest professor twice, first in 1981–1982 and again, exactly 10 years later, in 1991–1992. Both visits were immensely successful and laid the ground for continued cooperation during the following years. Dexter made numerous shorter visits to the department, and he served with great enthusiasm and competence on the advisory board for BRICS (Basic Research in Computer Science) Research Center and international PhD-School. BRICS covered both "Track A and B" activities (Algorithmics and Semantics), and as such Dexter was the ideal adviser. Looking back, we pay tribute to Dexter for his many contributions to computer science in Aarhus over a period of more than thirty years.

There can be no doubt that Dexter is among the most highly regarded theoretical computer scientists in the world. In addition, he is a devoted volunteer firefighter, an excellent (amateur) musician and he has an exceptional talent for learning languages. Three months after his first arrival in Aarhus (back in 1981) he announced that from now on he would speak only Danish when around the department. In the beginning this was a genuine "pain in $\cdots$", but after a couple of months he spoke Danish fluently (which he still does). So if visitors to foreign countries really want to learn the local language, this is definitely how to do it!

Erik first met Dexter in 1975 at Cornell, where they were both graduate students (and both with Juris Hartmanis as supervisor). Erik was with his family (wife and two children), and Dexter and Fran (who were already together then) were among the first to take care of this little Danish family making sure to include them in the social life around the department. We hope that we succeeded in repaying some of this hospitality during Dexter's second sabbatical, where his family now consisted of Fran, Alexander, Geoffrey and Timothy.

At the personal level, Dexter is always interesting to be around. He takes a genuine interest in other people, he engages enthusiastically in discussions and he has a well developed sense of humor. We hope that more that 30 years of cooperation and friendship will continue in the future, and who knows maybe another sabbatical in Aarhus (20 years later) will materialize?

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, p. 341, 2012.

Travelling with Dexter Kozen

Peter van Emde Boas

ILLC-FNWI-University of Amsterdam &
Bronstee.com Software & Services B.V.,
Franz Lisztlaan 5, 2012 CJ Heemstede, The Netherlands
peter@bronstee.com

Abstract. Aside from our shared interest in particular areas of Theoretical Computer Science, the experience I share with Dexter Kozen originates from the fact that we both belong to the not too large group of Theoreticiancs who during the late 1970-ies and early 1980-ies frequently crossed the Iron Curtain in order to interact with our colleagues from the Socialist part of Europe. This results into many appearances of Dexter in my collection of pictures from that period; a collection I like to share with the readers of this volume.

There is definitely a large overlap between the research Interests of Dexter Kozen and myself. Yet this has not resulted in any joint publication or a shared supervised student. The closest we came to a joint publication is the fact that I contributed a paper to the volume Dexter edited on the Logic of Programs workshop in Yorktown Heights in 1981 (item 161[1]), while Dexter contributed to the Festschrift on the occasion of the ph.d. Defence of Arjen K. Lenstra in 1984 which was edited by a group including myself (item 10).

What we do share is a large number of social encounters at various meetings, particularly in Eastern Europe at the time when the Iron Curtain still was a fact of life, a reality believed to persist for the oncoming decades. We both belonged to the not too large community of Theoretical Computer Scienctist who frequently traveled to the East in order to participate or to contribute as speaker or program committee member to the Theory conferences organised in these countries, like MFCS, FCT or the smaller workshops organised in Poznan or East Berlin. We have also had our mutual visits to our home institutes.

Starting the fall of 1976 I have been making pictures at almost all lectures and other events attended by me. The pictures involve speakers, but also session chairs and (occasionally) the audience. The collection also includes pictures made during the social events (diners and excursions). Today, given the availability of digital cameras with virtually unbounded storage capacity such pictures are quite common, but in the 1970-ies and 1980-ies we still were operating with a limited amount of film material, which moreover not always could be refreshed, particularly not in Eastern Europe.

[1] The numbers mentioned refer to a version of Dexter's CV dated August 23 2011 which I found on the Web.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 342–351, 2012.

The earliest pictures of Dexter in my collection were made in the USA rather than Europe. Our first recorded encounter must have been at the FOCS 1978 meeting in Ann Arbor where Dexter was presenting a paper on his Mice and Maze work (item 60).

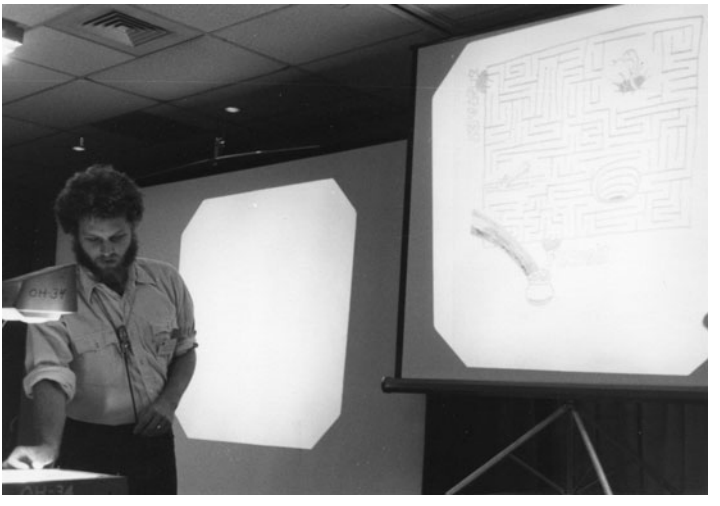


Fig. 1. FOCS 17, Oct 17 1987

In this presentation Dexter succeeded to give a single slide presentation of the rather difficult proof of Budach's theorem that Finite Automata can't escape from mazes.

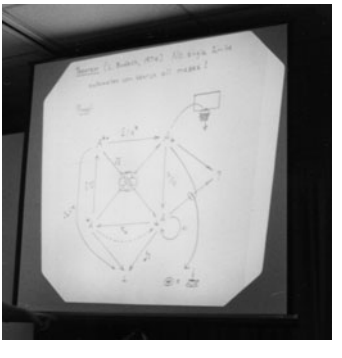


Fig. 2. Budach's proof

The next year we shared a trip to a workshop in Poznan organised by Marek Karpinski and his colleagues, followed by the FCT 1979 meeting in Wendisch Riesz in the GDR. Aside from Dexter the picture shows Marek Karpinski and his wife Jitka, and also my wife Ghica and my eldest son Donald appear.

Fig. 3. Poznan Sep 11 1979

The next picture shows some of the other participants of the workshop: A Salwicki, M. Chytil and mr. Habashinsky. The next two pictures were made at the workshop. Again Dexter was presenting his work on Mice and Mazes.

Fig. 4. Poznan Sep 11 1979

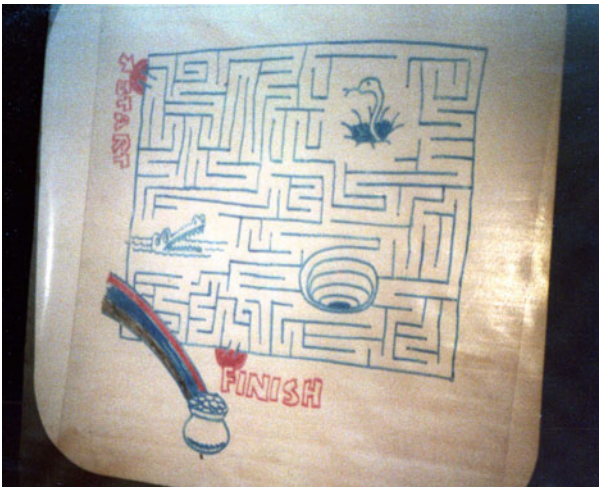


Fig. 5. Poznan Sep 12 1979

This picture shows the same slide as used in Ann Arbor the year before, but this time in full colour (provided you have access to the .pdf).

Fig. 6. Frankfurt am Oder, Sep 20 1979

On the next picture we encounter Dexter at the High table of the conference diner in Frankfurt am Oder; the table is chaired by Oberburgemeister Hasse; left of him is Lothar Budach.

The next picture was made in Noordwijkerhout on the occasion of the ICALP meeting in July 1980. To save expenses I had obtained permission of my colleage mrs. E Dobber to use her vacation house close to the conference site for housing a few

of our Eastern European representatives. Needless to say that we had a party there. The picture shows aside from Dexter also A. Goralcikova and P. Berman.

Fig. 7. Noordwijkerhout, Jul. 17 1980

Fig. 8. Garisson, May 03 1981

The next two pictures were made at the time of the Logic of Programs workshop in Yorktown Heights. The first one shows next to Dexter and Marek Karpinski also Lutz Priese, Jurek Tyurin and myself. The second one shows H Langmaack, D Luckham and W.P de Roever. No pictures were made at the workshop; no cameras were allowed within an IBM laboratory at that time.

Fig. 9. Garisson May 03 1981

Next we move to the ICALP in Aarhus in July 1982. Dexter has spent a Visiting professorate there and is hosting me as a visitor. Together with his wife Fran he shows me the local street sculptures.

Fig. 10. Aarhus July 08 1982

Fig. 11. Aarhus July 15 1982

Next his performance at the ICALP conference dinner where he is reading the Danish translation of Jabberwocky composed at the diner.

Fig. 12. Washington D.C., Apr 29 1984

Back to the USA. Before the start of STOC 1984 in Washington we had a small reception at the house of John Cherniavsky, where we meet Dexter discussing with Mike Paterson and Albert Meyer.

We skip a few years till 1987 when Dexter is organizing the IEEE Structure in Complexity Theory 2 and LICS 2 meetings in Cornell. We meet him with Fran and the two elder sons in Thauganook Park for one of the Social events, and we see how efficient he is carrying documents on a bike without additional equipment for transporting objects.

Fig. 13. Cornell Jun 18 1987

Fig. 14. Cornell Jun 22 1987

Fig. 15. Heemstede May 15 1992

Fig. 16. Amsterdam, May 15 1992

I continue with two pictures made again in the Netherlands. The first shows Dexter navigating the pond behind my house, and the second one where he presents one of the many talks given in Amsterdam.

One final example illustrating how Dexter has influenced my work. The completeness result on Kleene Algebras (item 83) which he presented at the Amsterdam LICS in 1991 is actually much older. I remember that he gave me a preprint of an earlier version sometime during the early 1980-ies, and that I decided that it was a marvelous result, so I presented it to my students in the third year's course on automata theory (it

must have occurred before 1985 since starting that year the course was transferred to other teachers). A nice mixture of formal language theory and algebra which has been an important ingredient in my automata course since 1973. It is questionable whether my students shared my taste for good mathematics at that occasion.

It remains to show what Dexter looks like today. The final picture shows Dexter during a talk given at the CWI on Nov 07 2011. Evidently he has aged, but not the quality of his work in Mathematics and Computer Science.

Fig. 17. Amsterdam Nov 07 2011

Dexter as a PhD Advisor

Brad Vander Zanden

University of Tennessee, Knoxville, TN 37920, USA
bvz@eecs.utk.edu

Dexter and I work in very different areas in Computer Science, so I cannot attest to his successes in Computer Science as I am sure others so eloquently will. However, I was Dexter's first PhD graduate, and I can certainly attest to his skills as a PhD advisor. Dexter actually arrived at Cornell after I started my PhD program. He had a buzz of electricity about him and that quality above all others initially attracted my interest. At that time Dexter was already an accomplished theoretician. I know that he has considerably burnished that already formidable reputation in the intervening decades. My story is a different one though, and it winds through the area of graphical user interfaces.

Graphical user interfaces are now well entrenched in computer science, but in the mid 1980s this field was just starting to develop. Somehow I had developed an interest in this area, but no one on the faculty at Cornell was doing research in GUIs. Then Dexter arrived, and while his primary interest seemed to be in the more theoretical aspects of programming languages, he did have a side-interest in designing a visual language for scientific computing. A group of us would have a weekly lunch meeting at Vinnies, a now defunct pizzeria in Ithaca, to discuss the design of the language. Dexter christened this language Alex, after his first son, and ultimately graduated a PhD student, Eugene Ressler, who more fully developed the language.

My interest progressed in a different direction, as it focused on adapting attribute grammars to the problem of formally specifying a GUI, and then developing a tool to take that specification and generate an actual GUI. Fortunately Dexter was accommodating and agreed to be my PhD advisor, even though this subject was pretty far afield from his area of expertise. It was only as I matured as a researcher that I truly realized the risk that Dexter took in taking me on as a PhD advisee. When you accept a PhD advisee who is outside your natural area, it is much more difficult to assess that person's contribution, as you are not familiar with the related work in that area. Considering that Dexter's realm tends toward the abstract, mathematical side of computer science, and my realm tends toward the concrete, and more psychological, side of computer science, Dexter was taking something of a leap. I am indebted to Dexter for taking this risk, as I would have had to otherwise leave Cornell and go elsewhere to pursue my interest.

One of Dexter's biggest contributions to my career was showing me how mathematics could be applied to GUIs in a beneficial manner. Keeping up with Dexter is not easy. I had a business background when I came to Cornell, and although I worked on beefing up my mathematical credentials, they never rivaled Dexter's mathematical wizardry. I remember entering Dexter's office to discuss my

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 352–353, 2012.

weekly progress and leaving an hour later with my head spinning from all the mathematics and ideas that Dexter ran through in that hour. I usually would go back to my office and spend some time talking with my officemate Bill Pugh, trying to work out the ins-and-outs of Dexter's insights. Over the months with Dexter I slowly improved my abilities to design and analyze algorithms, and to speak about them precisely. I still remember one line from my PhD dissertation, which initially started out as reading "An NP-complete problem requires an exponential amount of time to solve." To someone like me who was not really into theory, this statement was "close enough". Of course this statement made Dexter shutter and I got a nice little lecture about it. The line that was finally published read "It is commonly conjectured that an NP-complete problem requires an exponential amount of time to solve." This attention to precision has served me well over the years. On a number of occasions I was able to help an author in the GUI area tighten up their proof because of the experience that I gained under Dexter's tutelage.

Dexter also delivered some first-rate career advice, although I did not particularly appreciate his efforts at first. Towards the end of my dissertation, Dexter suggested that it might be a good idea to get a post-doc and work with someone more deeply associated with GUIs, before pursuing a position in academia. Being bull-headed, I pushed ahead with applying for academic positions. However, because I was not yet that well-known, that search did not fare particularly well. I ended up doing precisely what Dexter suggested and accepted a post-doc position at Carnegie Mellon with one of the leading GUI researchers, both then and now, Brad Myers. Brad really helped shepherd my career in GUI and when I emerged from CMU 2 years later, I was a rising star in the GUI field and had many more job opportunities. Dexter's intuition and nudge in this direction proved to be just the right tonic for my career.

I will always remember Dexter fondly for taking a chance on a student so far removed from his own normal area of expertise, particularly at such an early juncture in his own academic career. Despite the difference in areas, it speaks volumes about both his intellectual ability and his versatility that he was able to provide me with such invaluable guidance throughout my PhD dissertation. Thanks to his help, support, and wisdom, I was able to launch what has been a very satisfying and fulfilling academic career.

Rock'n'Roll Computer Science

Fritz Henglein

Department of Computer Science, University of Copenhagen (DIKU)
henglein@diku.dk

I am not a student, colleague or co-author of Dexter's, yet his person and his work have influenced and inspired me since my graduate student days, a time when Dexter already was a towering figure in the minds of students interested in theoretical computer science.

It started with my Master's essay in Artificial Intelligence[1]. An algebraic structure emerged, which showed itself to have been investigated before: Under the name of *Dynamic Algebra* — with Dexter's name prominently attached as founder and key contributor. Later, in a graduate seminar by Steve Fortune and Steve Mahaney I was drawn to giving a presentation on Alternating Turing Machines, pioneered by Dexter. They struck me not only as a powerful tool for connecting logic and complexity classes, but as profoundly enlightening: A new way of thinking about computation!

Then, meeting Dexter for the first time at LICS 87 in Ithaca, I was amazed to find the outgoing, friendly, generous and fun person that he is, a larger-than-life figure who is all over of the map of life, not just the one of computer science. I thoroughly enjoyed playing on the same team with him during the traditional Logic versus Computer Science soccer match. (He chose the CS side – Dexter being ambidextrous, LICS-wise, he had his pick.) He was a great team player: Moving into open space, conquering it, always playable — and at the same time a finisher. Not unlike the way he plays computer science.

I chanced upon a contribution he had in the EATCS bulletin in 1996 on regularity-preserving functions and thought it was just ... beautiful. I felt compelled to send him an email saying just that: Beautiful! I haven't done such a thing since (in hindsight, I should have).

In 1997, a clever technique I thought my student Michael Brandt and I had developed for coinductive axiomatization turned out to be — in a technically more challenging setting — developed in Dexter's seminal modal μ-calculus paper.

During my visit at Cornell in 1999, Dexter showed me Kleene Algebra with Tests (my bad for only knowing it superficially then). He stood at the blackboard and was in the middle of developing a proof. Suddenly he turned towards me, held up the chalk and asked: "How would you solve that?" Nothing sharpens one's attention like the adrenalin kick of being called to the blackboard! Later that afternoon Dexter asked if I wanted to join in on a hockey game. It was a fantastic visit. (I wimped out on the hockey game, though.)

[1] *On Backpropagation of constraints*, April 1984, Rutgers University.

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 354–355, 2012.

On his visits to Denmark I have to warn friends and colleagues that the private channel strategy of speaking Danish fails miserably with Dexter: He not only speaks Danish fluently, he also knows all those words that you can't find in a regular Danish dictionary ...

Studying his seminal axiomatization of regular expression equivalence – the equational theory of Kleene Algebras, which has Dexter's massive influence scribbled all over its development – we (my Ph.D. student Lasse Nielsen and I) found that it not only turns out to be technically superior to others (which Dexter points out), but there was something "right" beyond that about it: his proof system turned out to have an appealing interpretation as a natural functional programming language, a way of putting proofs into an electric outlet and *running* them. More recently I have been looking at languages for probabilistic processes, and there he popped up again: Dexter laying the foundation of semantics of probabilistic programming languages a whopping 30 years ago. I suspect I am not the only one who is amazed at – and maybe unaware of the full scope of – the enormous breadth, depth and impact of his work.

Dexter is one of very few people I can think of who is at home in – and transcends – both "discrete math" TCS, which may fall victim to a "it's all bits – and thus all about a big-O/complexity classes" tarpit of thinking; and "semantics" TCS, which may fall victim to the "it's all structure – and all about full abstraction" tarpit. I have always thought of Dexter's work as the incarnation of *rock'n'roll computer science*: not just conceptually and technically deep computer science, but computer science that *rocks*. And that was before I found out that Dexter does do rock'n'roll in a literal sense (not to mention goes bone-crushingly full-body in rugby matches), but I am no longer surprised about such revelations.

Author Index