EPTCS 191

Proceedings of the

Tenth International Workshop on

Fixed Points in Computer Science

Berlin, Germany, September 11-12, 2015

Edited by: Ralph Matthes and Matteo Mio

Published: 9th September 2015
DOI: 10.4204/EPTCS.191
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of the Tenth International Workshop on Fixed Points in Computer Science (FICS 2015) which took place on September 11th and 12th, 2015 in Berlin, Germany, as a satellite event of the conference Computer Science Logic (CSL 2015).

Fixed points play a fundamental role in several areas of computer science. They are used to justify (co)recursive definitions and associated reasoning techniques. The construction and properties of fixed points have been investigated in many different settings such as: design and implementation of programming languages, logics, verification, databases. The aim of this workshop is to provide a forum for researchers to present their results to those members of the computer science and logic communities who study or apply the theory of fixed points.

The editors thank all authors who submitted papers to FICS 2015 (successful or not), and the program committee members Ulrich Berger, Dietmar Berwanger, Filippo Bonchi, Venanzio Capretta, Krishnendu Chatterjee, Kaustuv Chaudhuri, Thomas Colcombet, Makoto Hamana, Radu Mardare, Henryk Michalewski, Andrzej Murawski, Alexandra Silva and Sam Staton for their work in selecting the 11 papers of this volume. Every submission was evaluated by three or four reviewers (we are thankful to all the external anonymous reviewers that were involved but refrain from listing them here). Some of the papers were re-reviewed after revision.

Apart from presentations of the accepted papers, we are delighted that FICS 2015 featured two invited talks: Bartek Klin on the decidability of certain infinite constraint satisfaction problems and James Worrell on the decidability of certain variants of the Skolem Problem for linear recurrence sequences. Many thanks to them for having accepted the invitation.

We could also offer the FICS'15 audience the two invited talks of the colocated annual meeting of the GI-Fachgruppe "Logik in der Informatik", given by Ulrich Schöpp and Michael Elberfeld. Thanks to them and the organizers of that meeting for making this possible.

Finally, we would like to express our deep gratitude to CSL 2015 for local organization and to EACSL and ANR ("Agence Nationale de la Recherche", France) for funding FICS 2015.

Ralph Matthes,
Matteo Mio

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. ii–ii, doi:10.4204/EPTCS.191.0

Topological Dynamics and Decidability of Infinite Constraint Satisfaction

Bartek Klin
Warsaw University

A group is called extremely amenable if every action of it on a compact space has a fixpoint. One example, shown by Pestov, is the automorphism group of the total order of rational numbers. This fact is used to establish the decidability of certain infinite constraint satisfaction problems, based on nominal sets due to Pitts.

This talk is roughly based on the paper [KKOT15].

References

[KKOT15] Bartek Klin, Eryk Kopczynski, Joanna Ochremiak, and Szymon Torunczyk. Locally finite constraint satisfaction problems. In *Procs. LICS 2015*, pages 475–486, 2015.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 1–1, doi:10.4204/EPTCS.191.1

Reachability Problems for Continuous Linear Dynamical Systems

James Worrell
Department of Computer Science, Oxford University, UK

This talk is about reachability problems for continuous-time linear dynamical systems. A central decision problem in this area is the Continuous Skolem Problem [BDJB10], which asks whether a real-valued function satisfying an ordinary linear differential equation has a zero. This can be seen as a continuous analog of the Skolem Problem for linear recurrence sequences [HHHK05], which asks whether the sequence satisfying a given recurrence has a zero term. For both the discrete and continuous versions of the Skolem Problem, decidability is open.

We show that the Continuous Skolem Problem lies at the heart of many natural verification questions on linear dynamical systems, such as continuous-time Markov chains and linear hybrid automata. We describe some recent work, done in collaboration with Chonev and Ouaknine [COW15a, COW15b], that uses results in transcendence theory and real algebraic geometry to obtain decidability for certain variants of the problem. In particular, we consider a bounded version of the Continuous Skolem Problem, corresponding to time-bounded reachability. We prove decidability of the bounded problem assuming Schanuel's conjecture, one of the main conjectures in transcendence theory. We describe some partial decidability results in the unbounded case and discuss mathematical obstacles to proving decidability of the Continuous Skolem Problem in full generality.

References

[BDJB10] Paul C. Bell, Jean-Charles Delvenne, Raphaël M. Jungers, and Vincent D. Blondel. The Continuous Skolem-Pisot Problem. *Theoretical Computer Science*, 411(40-42):3625–3634, 2010.

[COW15a] Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the decidability of the Bounded Continuous Skolem Problem. *CoRR*, abs/1506.00695, 2015.

[COW15b] Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the decidability of the continuous infinite zeros problem. *CoRR*, abs/1507.03632, 2015.

[HHHK05] V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's Problem – on the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 2–2, doi:10.4204/EPTCS.191.2

Dependent Inductive and Coinductive Types are Fibrational Dialgebras

Henning Basold
Radboud University, iCIS, Intelligent Systems
CWI, Amsterdam, The Netherlands
h.basold@cs.ru.nl

In this paper, I establish the categorical structure necessary to interpret dependent inductive and coinductive types. It is well-known that dependent type theories à la Martin-Löf can be interpreted using fibrations. Modern theorem provers, however, are based on more sophisticated type systems that allow the definition of powerful inductive dependent types (known as inductive families) and, somewhat limited, coinductive dependent types. I define a class of functors on fibrations and show how data type definitions correspond to initial and final dialgebras for these functors. This description is also a proposal of how coinductive types should be treated in type theories, as they appear here simply as dual of inductive types. Finally, I show how dependent data types correspond to algebras and coalgebras, and give the correspondence to dependent polynomial functors.

1 Introduction

It is a well-established fact that the semantics of inductive data types without term dependencies can be given by initial algebras, whereas the semantics of coinductive types can be given by final coalgebras. However, for types that depend on terms, the situation is not as clear-cut.

Partial answers for inductive types can be found in [3, 8, 9, 11, 14, 19, 20], where semantics have been given for inductive types through polynomial functors in the category of set families or in locally Cartesian closed categories. Similarly, semantics for non-dependent coinductive types have been given in [1, 2, 6] by using polynomial functors on locally Cartesian closed categories. Finally, an interpretation for Martin-Löf type theory (without recursive type definitions) has been given in [21] and corrected in [16].

So far, we are, however, lacking a full picture of dependent coinductive types that arise as duals of dependent inductive types. To actually get such a picture, I extend in the present work Hagino's idea [13], of using dialgebras to describe data types, to dependent types. This emphasises the actual structure behind (co)inductive types as their are used in systems like Agda.[1] Moreover, dialgebras allow for a direct interpretation of types in this categorical setup, without going through translations into, for example, polynomial functors.

Having defined the structures we need to interpret dependent data types, it is natural to ask whether this structure is actually sensible. The idea, pursued here, is that we want to obtain initial and final dialgebras from initial algebras and final coalgebras for polynomial functors. This is achieved by showing that the dialgebras in this work correspond to algebras and coalgebras, and that their fixed points can be constructed from fixed points of polynomial functors (in the sense of [12]).

[1] It should be noted that, for example, Coq treats coinductive types differently. In fact, the route taken in Agda with copatterns and in this work is much better behaved.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 3–17, doi:10.4204/EPTCS.191.3

To summarise, this paper makes the following contributions. First, we get a precise description of the categorical structure necessary to interpret inductive and coinductive data types, which can be seen as categorical semantics for an extension of the inductive and (copattern-based) coinductive types of Agda. The second contribution is a reduction to fixed points of polynomial functors.

What has been left out, because of space constraints, is an analysis of the structures needed to obtain induction and coinduction principles. Moreover, to be able to get a sound interpretation, with respect to type equality of dependent types, we need to require a Beck-Chevalley condition. This condition can be formulated for general (co)inductive types, but is also not given here.

Related work As already mentioned, there is an enormous body of work on obtaining semantics for (dependent) inductive, and to some extent, coinductive types, see [3, 11, 14, 20]. In the present work, we will mostly draw from [2] and [12]. Categorical semantics for basic Martin-Löf type theory have been developed, for example, in [16]. An interpretation, closer to the present work, is given in terms of fibrations by Jacobs [17]. In the first part of the paper, we develop everything on rather arbitrary fibrations, which makes the involved structure more apparent. Only in the second part, where we reduce data types to polynomial functors, we will work with slice categories, since most of the work on polynomial functors in that setting [2, 12]. Last, but not least, the starting idea of this paper is of course inspired by the dialgebras of Hagino [13]. These have also been applied to give semantics to induction-induction [4] schemes.

Outline The rest of the paper is structured as follows. In Section 2, we analyse a typical example of a dependent inductive type, namely vectors, that is, lists indexed by their length. We develop from this example a description of inductive and coinductive dependent data types in terms of dialgebras in fibrations. This leads to the requirements on a fibration, given in Section 3, that allow the interpretation of data types. In the same section, we show how dependent and fibrewise (co)products arise canonically in such a structure, and we give an example of a coinductive type (partial streams) that can only be treated in Agda through a cumbersome encoding. The reduction of dependent data types to polynomial functors is carried out in Section 4, and finish with concluding remarks in Section 5.

Acknowledgement I would like to thank the anonymous reviewers, who gave very valuable feedback and pointed me to some more literature.

2 Fibrations and Dependent Data Types

In this section we introduce *dependent data types* as initial and final dialgebras of certain functors on fibres of fibrations. We go through this setup step by step.

Let us start with dialgebras and their homomorphisms.

Definition 2.1. Let $\mathbf{C}$ and $\mathbf{D}$ be categories and $F, G : \mathbf{C} \to \mathbf{D}$ functors. An (F,G)-*dialgebra* is a morphism $c : FA \to GA$ in $\mathbf{D}$, where A is an object in $\mathbf{C}$. Given dialgebras $c : FA \to GA$ and $d : FB \to GB$, a morphism $h : A \to B$ is said to be a (dialgebra) *homomorphism* from c to d, if $Gh \circ c = d \circ Fh$. This allows us to form a category $\mathrm{DiAlg}\,(F,G)$, in which objects are pairs (A,c) with $A \in \mathbf{C}$ and $c : FA \to GA$, and morphisms are dialgebra homomorphisms.

The following example shows that dialgebras arise naturally from data types.

Example 2.2. Let A be a set, we denote by A^n the n-fold product of A, that is, lists of length n. Vectors over A are given by the set family $\mathrm{Vec}\,A = \{A^n\}_{n\in\mathbb{N}}$, which is an object in the category $\mathbf{Set}^{\mathbb{N}}$ of families

indexed by $\mathbb{N}$. In general, this category is given for a set I by

$$\mathbf{Set}^I = \begin{cases} \text{objects} & X = \{X_i\}_{i\in I} \\ \text{morphisms} & f = \{f_i : X_i \to Y_i\}_{i\in I} \end{cases}.$$

Vectors come with two constructors: $\text{nil} : \mathbf{1} \to A^0$ for the empty vector and prefixing $\text{cons}_n : A \times A^n \to A^{n+1}$ of vectors with elements of A. We note that $\text{nil} : \{\mathbf{1}\} \to \{A^0\}$ is a morphism in the category $\mathbf{Set}^{\mathbf{1}}$ of families indexed by the one-element set $\mathbf{1}$, whereas $\text{cons} = \{\text{cons}_n\} : \{A \times A^n\}_{n\in\mathbb{N}} \to \{A^{n+1}\}_{n\in\mathbb{N}}$ is a morphism in $\mathbf{Set}^{\mathbb{N}}$.

Let $F, G : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbf{1}} \times \mathbf{Set}^{\mathbb{N}}$ be the functors into the product of $\mathbf{Set}^{\mathbf{1}}$ and $\mathbf{Set}^{\mathbb{N}}$ with

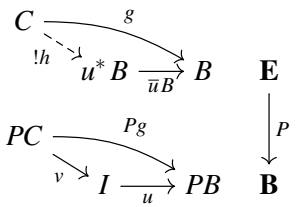
$$F(X) = (\{\mathbf{1}\}, \{A \times X_n\}_{n\in\mathbb{N}}) \qquad G(X) = (\{X_0\}, \{X_{n+1}\}_{n\in\mathbb{N}}).$$

Using these, we find that $(\text{nil}, \text{cons}) : F(\text{Vec}A) \to G(\text{Vec}A)$ is an (F,G)-dialgebra, in fact, it is the *initial* (F,G)-dialgebra.

Definition 2.3. An (F,G)-dialgebra $c : FA \to GA$ is called *initial*, if for every (F,G)-dialgebra $d : FB \to GB$ there is a unique homomorphism h from c to d, the *inductive extension* of d. Dually, (A,c) is *final*, provided there is a unique homomorphism h from any other dialgebra (B,d) into c. Here, h is the *coinductive extension* of d.

Having found the algebraic structure underlying vectors, we continue by exploring how we can handle the change of indices in the constructors. It turns out that this is most conveniently done by using fibrations.

Definition 2.4. Let $P : \mathbf{E} \to \mathbf{B}$ be a functor, where the $\mathbf{E}$ is called the *total* category and $\mathbf{B}$ the *base* category. A morphism $f : A \to B$ in $\mathbf{E}$ is said to be *cartesian over* $u : I \to J$, provided that i) $Pf = u$, and ii) for all $g : C \to B$ in $\mathbf{E}$ and $v : PC \to I$ with $Pg = u \circ v$ there is a unique $h : C \to A$ such that $f \circ h = g$. For P to be a *fibration*, we require that for every $B \in \mathbf{E}$ and $u : I \to PB$ in $\mathbf{B}$, there is a cartesian morphism $f : A \to B$ over u. Finally, a fibration is *cloven*, if it comes with a unique choice for A and f, in which case we denote A by u^*B and f by $\overline{u}B$, as displayed in the diagram on the right.



At first sight, this definition is arguably intimidating to someone who has never been exposed to fibrations. The idea is that the base category $\mathbf{B}$ contains as objects the indices of objects in $\mathbf{E}$, and as morphisms substitutions. The result of carrying out a substitution on indices, is captured by the Cartesian lifting property. Let us illustrate this on set families. We define $\text{Fam}(\mathbf{Set})$ to be the category

$$\text{Fam}(\mathbf{Set}) = \begin{cases} \text{objects} & (I, X : I \to \mathbf{Set}),\ I \text{ a set} \\ \text{morphisms} & (u,f) : (I,X) \to (J,Y) \text{ with } u : I \to J \text{ and } \{f_i : X_i \to Y_{u(i)}\}_{i\in I} \end{cases}$$

in which composition is defined by

$$(v,g) \circ (u,f) = \left(v \circ u, \{X_i \xrightarrow{f_i} Y_{u(i)} \xrightarrow{g_{u(i)}} Z_{v(u(i))}\}_{i\in I}\right).$$

A concrete object is the pair $(\mathbb{N}, \text{Vec}A)$, where $\text{Vec}A$ is the family of vectors from Ex. 2.2.

We define a cloven fibration on set families. Let $P : \text{Fam}(\mathbf{Set}) \to \mathbf{Set}$ be the projection on the first component, that is, $P(I,X) = I$ and $P(u,f) = u$. For a family (J,Y) and a function $u : I \to J$, we define

$u^*Y = \{Y_{u(i)}\}_{i\in I}$ and $\overline{u}Y = \big(u, \{\mathrm{id} : Y_{u(i)} \to Y_{u(i)}\}_{i\in I}\big)$. Then, for each $(w,g) : (K,Z) \to (J,Y)$ and $v : K \to I$ with $w = u \circ v$, we can define the morphism $(K,Z) \to (I,u^*Y)$ to be (v,h) with $h_k : Z_k \to Y_{u(v(k))}$ and $h_k = g_k$, since $u(v(k)) = w(k)$.

An important concept is the *fibre above* an object $I \in \mathbf{B}$, given by the category

$$\mathbf{P}_I = \begin{cases} \text{objects} & A \in \mathbf{E} \text{ with } P(A) = I \\ \text{morphisms} & f : A \to B \text{ with } P(f) = \mathrm{id}_I \end{cases}.$$

In a cloven fibration, we can use the Cartesian lifting to define for each $u : I \to J$ in $\mathbf{B}$ a functor $u^* : \mathbf{P}_J \to \mathbf{P}_I$, together with natural isomorphisms $\mathrm{Id}_{\mathbf{P}_I} \cong \mathrm{id}_I^*$ and $u^* \circ v^* \cong (v \circ u)^*$, see [17, Sec. 1.4]. The functor u^* is called *reindexing* along u.

Assumption 2.5. We assume all fibrations to be cloven in this work.

We are now in the position to take a more abstract look at our initial example.

Example 2.6. First, we note that the fibre of $\mathrm{Fam}(\mathbf{Set})$ above I is isomorphic to $\mathbf{Set}^I$. Let then $z : \mathbf{1} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ be $z(*) = 0$ and $s(n) = n+1$, giving us reindexing functors $z^* : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbf{1}}$ and $s^* : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$. By their definition, $z^*(X) = \{X_0\}$ and $s^*(X) = \{X_{n+1}\}_{n\in\mathbb{N}}$, hence the functor G, we used to describe vectors as dialgebra, is $G = \langle z^*, s^*\rangle$. In Sec. 3, we address the structure of F.

We generalise this situation to account for arbitrary data types.

Definition 2.7. Let $P : \mathbf{E} \to \mathbf{B}$ be a fibration. A *(dependent) data type signature*, parameterised by a category $\mathbf{C}$, is a pair (F,u) consisting of

- a functor $F : \mathbf{C} \times \mathbf{P}_I \to \mathbf{D}$ with $\mathbf{D} = \prod_{k=1}^n \mathbf{P}_{J_k}$ for some $n \in \mathbb{N}$ and $J_k, I \in \mathbf{B}$, and
- a family u of n morphisms in $\mathbf{B}$ with $u_k : J_k \to I$ for $k = 1,\dots,n$.

A family u as above induces a functor $\langle u_1^*,\dots,u_n^*\rangle : \mathbf{P}_I \to \mathbf{D}$, which we will often denote by G_u. This will enable us to define data types for such signatures, but let us first look at an example for the case $\mathbf{C} = \mathbf{1}$, that is, if $F : \mathbf{P}_I \to \mathbf{D}$ is not parameterised.

Example 2.8. A fibration $P : \mathbf{E} \to \mathbf{B}$ is said to have dependent coproducts and products, if for each $f : I \to J$ in $\mathbf{B}$ there are functors $\coprod_f$ and $\prod_f$ from $\mathbf{P}_I$ to $\mathbf{P}_J$ that are respectively left and right adjoint to f^*. For each $X \in \mathbf{P}_I$, we can define a signature, such that $\coprod_f(X)$ and $\prod_f(X)$ arise as data types for these signatures, as follows. Define the constant functor

$$K_X : \mathbf{P}_J \to \mathbf{P}_I \qquad K_X(Y) = X \qquad K_X(g) = \mathrm{id}_X\,.$$

Then (K_X, f) is the signature for coproducts and products. For example, the unit η of the adjunction $\coprod_f \dashv f^*$ will be the initial (K_X, f^*)-dialgebra $\eta_X : K_X(\coprod_f(X)) \to f^*(\coprod_f(X))$, using that $K_X(\coprod_f(X)) = X$. We come back to this in Ex. 2.10. □

To define data types in general, we allow them to have additional parameters, that is, we allow signatures (F,u), where $F : \mathbf{C} \times \mathbf{P}_I \to \mathbf{D}$ and $\mathbf{C}$ is a non-trivial category. Let us first fix some notation. We put $F(V,-)(X) = F(V,X)$ for $V \in \mathbf{C}$, which is a functor $\mathbf{P}_I \to \mathbf{D}$. Assume that the initial $(F(V,-), G_u)$-dialgebra $\alpha_V : F(V,\Phi_V) \to G_u(\Phi_V)$ and final $(G_u, F(V,-))$-dialgebra $\xi_V : G_u(\Omega_V) \to F(V,\Omega_V)$ exist. Then we can define functors $\mu(\widehat{F},\widehat{G_u}) : \mathbf{C} \to \mathbf{P}_I$ and $\nu(\widehat{G_u},\widehat{F}) : \mathbf{C} \to \mathbf{P}_I$, analogous to [18], by

$$\mu(\widehat{F},\widehat{G_u})(V) = \Phi_V \qquad \mu(\widehat{F},\widehat{G_u})(f : V \to W) = \big(\alpha_W \circ F(f, \mathrm{id}_{\Phi_W})\big)^-$$
$$\nu(\widehat{G_u},\widehat{F})(V) = \Omega_V \qquad \nu(\widehat{G_u},\widehat{F})(f : V \to W) = \big(F(f, \mathrm{id}_{\Omega_V}) \circ \xi_V\big)^\sim,$$

where the bar and tilde superscripts denote the inductive and coinductive extensions, that is, the unique homomorphism given by initiality and finality, respectively. The reason for the notation $\mu(\widehat{F},\widehat{G_u})$ and $\nu(\widehat{G_u},\widehat{F})$ is that these are initial and final dialgebras for the functors

$$\widehat{F},\widehat{G_u}:[\mathbf{C},\mathbf{P}_I]\to[\mathbf{C},\mathbf{D}] \qquad \widehat{F}(H)=F\circ\langle \mathrm{Id}_{\mathbf{C}},H\rangle \qquad \widehat{G_u}(H)=G_u\circ H$$

on functor categories. That the families α_V and ξ_V are natural in V follows directly from the definition of the functorial action as (co)inductive extensions. Hence, they give rise to dialgebras $\alpha\colon \widehat{F}(\mu(\widehat{F},\widehat{G_u}))\Rightarrow \widehat{G_u}(\mu(\widehat{F},\widehat{G_u}))$ and $\xi\colon \widehat{G_u}(\nu(\widehat{G_u},\widehat{F}))\Rightarrow \widehat{F}(\nu(\widehat{G_u},\widehat{F}))$.

Definition 2.9. Let (F,u) be a data type signature. An *inductive data type* (IDT) for (F,u) is an initial $(\widehat{F},\widehat{G_u})$-dialgebra with carrier $\mu(\widehat{F},\widehat{G_u})$. Dually, a coinductive data type (CDT) for (F,u) is a final $(\widehat{G_u},\widehat{F})$-dialgebra, note the order, with the carrier being denoted by $\nu(\widehat{G_u},\widehat{F})$. If $\mathbf{C}=\mathbf{1}$, we drop the hats from the notation.

Example 2.10. We turn the definition of the product and coproduct from Ex. 2.8 into actual functors. The observation we use is that the projection functor $\pi_1:\mathbf{P}_I\times\mathbf{P}_J\to\mathbf{P}_I$ gives us a "parameterised" constant functor: $K^J_A=\pi_1(A,-)$. If we are given $f:I\to J$ in $\mathbf{B}$, then we use the signature (π_1,f), and define $\coprod_f=\mu(\widehat{\pi_1},\widehat{f^*})$ and $\prod_f=\nu(\widehat{f^*},\widehat{\pi_1})$. We check the details of this definition in Thm. 3.2.

3 Data Type Completeness

We now define a class of signatures and functors that should be seen as categorical language for, what is usually called, strictly positive types [3], positive generalised abstract data types [14] or descriptions [8, 9]. Note, however, that none of these treat coinductive types. A *non-dependent* version of strictly positive types that include coinductive types are given in [2].

Let us first introduce some notation. Given categories $\mathbf{C}_1$ and $\mathbf{C}_2$ and an object $A\in\mathbf{C}_1$, we denote by $K^{\mathbf{C}_1}_A:\mathbf{C}_1\to\mathbf{C}_2$ the functor mapping constantly to A. The projections on product categories are denoted, as usual, by $\pi_k:\mathbf{C}_1\times\mathbf{C}_2\to\mathbf{C}_k$. Using these notations, we can define what we understand to be a data type by mutual induction.

Definition 3.1. A fibration $P:\mathbf{E}\to\mathbf{B}$ is *data type complete*, if all IDTs and CDTs for *strictly positive signatures* $(F,u)\in\mathscr{S}$ exist, where $\mathscr{S}$ is given by the following rule.

$$\frac{\mathbf{D}=\prod_{i=1}^n\mathbf{P}_{J_i} \qquad F\in\mathscr{D}_{\mathbf{C}\times\mathbf{P}_I\to\mathbf{D}} \qquad u=(u_1:J_1\to I,\dots,u_n:J_n\to I)}{(F,u)\in\mathscr{S}_{\mathbf{C}\times\mathbf{P}_I\to\mathbf{D}}}$$

The functors in $\mathscr{D}$ are given by the following rules, assuming that P is data type complete.

$$\frac{A\in\mathbf{P}_J}{K^{\mathbf{P}_I}_A\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_J}} \qquad \frac{\mathbf{C}=\prod_{i=1}^n\mathbf{P}_{I_i}}{\pi_k\in\mathscr{D}_{\mathbf{C}\to\mathbf{P}_{I_k}}} \qquad \frac{f:J\to I\text{ in }\mathbf{B}}{f^*\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_J}} \qquad \frac{F_1\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_K} \qquad F_2\in\mathscr{D}_{\mathbf{P}_K\to\mathbf{P}_J}}{F_2\circ F_1\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_J}}$$

$$\frac{F_i\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_{J_i}} \qquad i=1,2}{\langle F_1,F_2\rangle\in\mathscr{D}_{\mathbf{P}_I\to\mathbf{P}_{J_1}\times\mathbf{P}_{J_2}}} \qquad \frac{(F,u)\in\mathscr{S}_{\mathbf{C}\times\mathbf{P}_I\to\mathbf{D}}}{\mu(\widehat{F},\widehat{G_u})\in\mathscr{D}_{\mathbf{C}\to\mathbf{P}_I}} \qquad \frac{(F,u)\in\mathscr{S}_{\mathbf{C}\times\mathbf{P}_I\to\mathbf{D}}}{\nu(\widehat{G_u},\widehat{F})\in\mathscr{D}_{\mathbf{C}\to\mathbf{P}_I}}$$

This mutual induction is well-defined, as it can be stratified in the nesting of fixed points.

As a first sanity check, we show that a data type complete fibration has, both, fibrewise and dependent (co)products. These are instances of the following, more general, result.

Theorem 3.2. *Suppose $P : \mathbf{E} \to \mathbf{B}$ is a data type complete fibration. Let $\mathbf{C} = \prod_{i=1}^{m} \mathbf{P}_{K_i}$ and $\pi_1 : \mathbf{C} \times \mathbf{P}_I \to \mathbf{C}$ be the first projection. If $G_u : \mathbf{P}_I \to \mathbf{C}$ is such that (π_1, u) is a signature, then we have the following adjoint situation:*

$$\mu(\widehat{\pi_1}, \widehat{G_u}) \dashv G_u \dashv \nu(\widehat{G_u}, \widehat{\pi_1}).$$

Proof. We only show how the adjoint transposes are obtained in the case of inductive types. Concretely, for a tuple $V \in \mathbf{C}$ and an object $A \in \mathbf{P}_I$, we need to prove the correspondence

$$\frac{f: \quad \mu(\widehat{\pi_1}, \widehat{G_u})(V) \longrightarrow A \qquad \text{in } \mathbf{P}_I}{g: \qquad\qquad\qquad V \longrightarrow G_u A \quad \text{in } \mathbf{C}}$$

Let us use the notation $H = \mu(\widehat{\pi_1}, \widehat{G_u})$, then the choice of π_1 implies that the initial $(\widehat{\pi_1}, \widehat{G_u})$-dialgebra is of type $\alpha\colon \mathrm{Id}_{\mathbf{C}} \Rightarrow G_u \circ H$, since $\widehat{\pi_1}(H) = \pi_1 \circ \langle \mathrm{Id}_{\mathbf{C}}, H \rangle = \mathrm{Id}_{\mathbf{C}}$ and $\widehat{G_u}(H) = G_u \circ H$. This allows us to use as transpose of f the morphism $V \xrightarrow{\alpha_V} G_u(H(V)) \xrightarrow{G_u f} G_u A$. As transpose of g, we use the inductive extension of $\widehat{\pi_1}(K_A^{\mathbf{C}})(V) = V \xrightarrow{g} G_u A = \widehat{G_u}(K_A^{\mathbf{C}})(V)$. The proof that this correspondence is natural and bijective follows straightforwardly from initiality. For coinductive types, the result is given by duality. □

This gives fibrewise coproducts by $+_I = \mu(\widehat{\pi_1}, \widehat{G_u})$ and products by $\times_I = \nu(\widehat{G_u}, \widehat{\pi_1})$, using $u = (\mathrm{id}_I, \mathrm{id}_I)$. Dependent (co)products along $f : I \to J$ use $u = f$, see Ex. 2.10.

There are many more examples of data types that exist in a data type complete fibration. We describe three fundamental ones.

Example 3.3.

1. The first example are initial and final objects inside the fibres $\mathbf{P}_I$. Since an initial object is characterised by having a unique morphism *to* every other object, we define it as an initial dialgebra, namely $\mathbf{0}_I = \mu(\mathrm{Id}, \mathrm{id}_I^*)$. Then there is, for each $A \in \mathbf{P}_I$, a unique morphism $!^A : \mathbf{0}_I \to A$ given as inductive extension of id_A. Dually, we define the terminal object $\mathbf{1}_I$ in $\mathbf{P}_I$ to be $\nu(\mathrm{id}_I^*, \mathrm{Id})$ and for each A the corresponding unique morphism $!_A : A \to \mathbf{1}_I$ as the coinductive extension of id_A.

   Note that this also follows from Thm. 3.2, if we require that (co)inductive data types also exist if $\mathbf{C} = \mathbf{1}$ (the empty product) and $u = \{\}$ (empty family of morphisms). This allows us to define the initial and final object as functors $\mathbf{1} \to \mathbf{P}_I$.

2. There are several definable notions of equality, provided that $\mathbf{B}$ has binary products. A generic one is propositional equality $\mathrm{Eq} : \mathbf{P}_I \to \mathbf{P}_{I \times I}$, the left adjoint to the contraction functor $\delta^* : \mathbf{P}_{I \times I} \to \mathbf{P}_I$, which is induced by the diagonal $\delta : I \to I \times I$. Thus it is given by the dependent coproduct $\mathrm{Eq} = \coprod_\delta$ and the constructor $\mathrm{refl}_X : X \to \delta^*(\mathrm{Eq} X)$.

3. Assume that there is an object A^ω in $\mathbf{B}$ of streams over A, together with projections to head and tail. Then we can define bisimilarity between streams as CDT for the signature

$$F, G_u : \mathbf{P}_{(A^\omega)^2} \to \mathbf{P}_{(A^\omega)^2} \times \mathbf{P}_{(A^\omega)^2}$$
$$F = \left\langle (\mathrm{hd} \times \mathrm{hd})^* \circ K_{\mathrm{Eq}(A)}, (\mathrm{tl} \times \mathrm{tl})^* \right\rangle \qquad \text{and} \qquad u = (\mathrm{id}_{A^\omega \times A^\omega}, \mathrm{id}_{A^\omega \times A^\omega}).$$

   Note that there is a category $\mathrm{Rel}(\mathbf{E})$ of binary relations in $\mathbf{E}$ by forming the pullback of P along $\Delta : \mathbf{B} \to \mathbf{B}$ with $\Delta(I) = I \times I$, see [15]. Then we can reinterpret F and G_u by

$$F, G_u : \mathrm{Rel}(\mathbf{E})_{A^\omega} \to \mathrm{Rel}(\mathbf{E})_{A^\omega} \times \mathrm{Rel}(\mathbf{E})_{A^\omega}$$
$$F = \langle \mathrm{hd}^\# \circ K_{\mathrm{Eq}(A)}, \mathrm{tl}^\# \rangle \qquad \text{and} \qquad G_u = \langle \mathrm{id}_{A^\omega}^\#, \mathrm{id}_{A^\omega}^\# \rangle,$$

where $(-)^{\#}$ is reindexing in $\mathrm{Rel}(\mathbf{E})$. The final (G_u, F)-dialgebra is a pair of morphisms

$$(\mathrm{hd}^{\sim}_A : \mathrm{Bisim}_A \to \mathrm{hd}^{\#}(\mathrm{Eq}(A)), \mathrm{tl}^{\sim}_A : \mathrm{Bisim}_A \to \mathrm{tl}^{\#}(\mathrm{Bisim}_A)).$$

Bisim_A should be thought of to consist of all bisimilarity proofs. Coinductive extensions yield the usual coinduction proof principle, allowing us to prove bisimilarity by establishing a bisimulation relation $R \in \mathrm{Rel}(\mathbf{E})_{A^\omega}$ together with $h : R \to \mathrm{hd}^{\#}(\mathrm{Eq}(A))$ and $t : R \to \mathrm{tl}^{\#}(R)$, saying that the heads of related streams are equal and that the tails of related streams are again related.

The last example, we give, shall illustrate the additional capabilities of CDTs in the present setup over those currently available in Agda. However, one should note that coinductive types in Agda provide extra power in the sense that destructors can refer to each other. This is equivalent to having a strong coproduct [17, Sec. 10.1 and Def. 10.5.2], which we do not require in the setup of this work and thus A proof of this equivalence is left out because of space constraints.

Example 3.4. A partial stream is a stream together with a, possibly infinite, depth up to which it is defined. Assume that there is an object $\mathbb{N}^\infty$ of natural numbers extended with infinity and a successor map $s_\infty : \mathbb{N}^\infty \to \mathbb{N}^\infty$ in $\mathbf{B}$, we will see how these can be defined below. Then partial streams correspond to the following type declaration.

codata PStr (A : **Set**) : $\mathbb{N}^\infty \to$ **Set where**
  hd : (n : $\mathbb{N}^\infty$) $\to$ PStr (s_∞ n) $\to$ A
  tl : (n : $\mathbb{N}^\infty$) $\to$ PStr (s_∞ n) $\to$ PStr n

In an explicit, set-theoretic notation, we can define them as a family indexed by $n \in \mathbb{N}^\infty$:

$$\mathrm{PStr}(A)_n = \{s : \mathbb{N} \rightharpoonup A \mid \forall k < n. k \in \mathrm{dom}\, s \wedge \forall k \geq n. k \notin \mathrm{dom}\, s\},$$

where the order on $\mathbb{N}^\infty$ is given by extending that of the natural numbers with ∞ as strict top element, i.e., such that $k < \infty$ for all $k \in \mathbb{N}$.

The interpretation of $\mathrm{PStr}(A)$ for $A \in \mathbf{P_1}$ in a data type complete fibration is given, similarly to vectors, as the carrier of the final (G_u, F)-dialgebra, where

$$G_u, F : \mathbf{P}_{\mathbb{N}^\infty} \to \mathbf{P}_{\mathbb{N}^\infty} \times \mathbf{P}_{\mathbb{N}^\infty} \qquad G_u = \langle s^*_\infty, s^*_\infty \rangle \qquad F = \left\langle K^{\mathbb{N}^\infty}_{\overline{A}}, \mathrm{Id} \right\rangle$$

and $\overline{A} = !^*_{\mathbb{N}^\infty}(A) \in \mathbf{P}_{\mathbb{N}^\infty}$ is the weakening of A using $!_{\mathbb{N}^\infty} : \mathbb{N}^\infty \to \mathbf{1}$. The idea of this signature is that the head and tail of partial streams are defined only on those partial streams that are defined in, at least, the first position. On set families, partial streams are given by the dialgebra $\xi = (\mathrm{hd}, \mathrm{tl})$ with $\mathrm{hd}_n : \mathrm{PStr}(A)_{(s_\infty n)} \to A$ and $\mathrm{tl}_n : \mathrm{PStr}(A)_{(s_\infty n)} \to \mathrm{PStr}(A)_n$ for every $n \in \mathbb{N}^\infty$.

We can make this construction functorial in A, using the same "trick" as for sums and products. To this end, we define the functor $H : \mathbf{P_1} \times \mathbf{P}_{\mathbb{N}^\infty} \to \mathbf{P}_{\mathbb{N}^\infty} \times \mathbf{P}_{\mathbb{N}^\infty}$ with $H = \langle !_{\mathbb{N}^\infty} \circ \pi_1, \pi_2 \rangle$, where π_1 and π_2 are corresponding projection functors, so that $H(A, X) = F(X)$. This gives, by data type completeness, rise to a functor $\nu(\widehat{G_u}, \widehat{F}) : \mathbf{P}_{\mathbb{N}^\infty} \to \mathbf{P}_{\mathbb{N}^\infty}$, which we denote by PStr, together with a pair $(\mathrm{hd}, \mathrm{tl})$ of natural transformations. □

We have seen in the examples above that we would often like to use a data type again as index, which means that we need a mechanism to turn a data type in $\mathbf{E}$ into an index in $\mathbf{B}$. This is provided by, so called, *comprehension*.

Definition 3.5 (See [17, Lem. 1.8.8, Def. 10.4.7] and [10])**.** Let $P : \mathbf{E} \to \mathbf{B}$ be a fibration. If each fibre $\mathbf{P}_I$ has a final object $\mathbf{1}_I$ and these are preserved by reindexing, then there is a fibred *final object functor* $\mathbf{1}_{(-)} : \mathbf{B} \to \mathbf{E}$. (Note that then $P(\mathbf{1}_I) = I$.) P is a *comprehension category with unit* (CCU), if $\mathbf{1}_{(-)}$ has a right adjoint $\{-\} : \mathbf{E} \to \mathbf{B}$, the *comprehension*. This gives rise to a functor $\mathscr{P} : \mathbf{E} \to \mathbf{B}^{\to}$ into the arrow category over $\mathbf{B}$, by mapping $A \mapsto P(\varepsilon_A) : \{A\} \to P(A)$, where $\varepsilon : \mathbf{1}_{\{-\}} \Rightarrow \mathrm{Id}$ is the counit of $\mathbf{1}_{(-)} \dashv \{-\}$. We often denote $\mathscr{P}(A)$ by π_A and call it the *projection* of A. Finally, P is said to be a *full* CCU, if $\mathscr{P}$ is full.

Note that, in a data type complete category, we can define final objects in each fibre, the preservation of them needs to be required separately.

Example 3.6. In Fam(**Set**), the final object functor is given by $\mathbf{1}_I = (I, \{\mathbf{1}\}_{i\in I})$, where $\mathbf{1}$ is the singleton set. Comprehension is defined to be $\{(I,X)\} = \coprod_{i\in I} X_i$ and the projections π_I map then an element of $\coprod_{i\in I} X_i$ to its component $i \in I$.

Using comprehension, we can give a general account to dependent data types.

Definition 3.7. We say that a fibration $P : \mathbf{E} \to \mathbf{B}$ is a *data type closed category* (DTCC), if it is a CCU, has a terminal object in $\mathbf{B}$ and is data type complete.

As already mentioned, the purpose of introducing comprehension is that it allows us to use data types defined in $\mathbf{E}$ again as index. The terminal object in $\mathbf{B}$ is used to introduce data types without dependencies, like the natural numbers. Let us reiterate on Ex. 3.4.

Example 3.8. Recall that we assumed the existence of extended naturals $\mathbb{N}^\infty$ and the successor map s_∞ on them to define partial streams. We are now in the position to define, in a data type closed category, everything from scratch as follows.

Having defined $+ : \mathbf{P_1} \times \mathbf{P_1} \to \mathbf{P_1}$, see Thm. 3.2, we put $\mathbb{N}^\infty = \nu(\mathrm{Id}, \mathbf{1} + \mathrm{Id})$ and find the predecessor pred as the final dialgebra on $\mathbb{N}^\infty$. The successor s_∞ arises as the coinductive extension $(\mathbb{N}^\infty, \kappa_2) \to (\mathbb{N}^\infty, \mathrm{pred})$, where κ_2 is the coproduct inclusion. Partial streams $\mathrm{PStr} : \mathbf{P}_{\{\mathbb{N}^\infty\}} \to \mathbf{P}_{\{\mathbb{N}^\infty\}}$ are then given, as in Ex. 3.4, by the final $(\widehat{G}, \widehat{F})$-dialgebra with $G = \langle \{s_\infty\}^*, \{s_\infty\}^* \rangle$ and $F = \langle !_{\mathbb{N}^\infty} \circ \pi_1, \pi_2 \rangle$. □

4 Constructing Data Types

In this section, we show how some data types can be constructed through polynomial functors, where I draw from the vast amount of work on polynomial functors that exists in the literature, see [2, 12]. The construction works by, first, reducing dialgebras to (co)algebras and, second, constructing the necessary initial algebras and final coalgebras as fixed points of polynomial functors analogously to the construction of strictly positive types in [2]. This result works thus far only for data types that, if at all, only use dependent coinductive types at the top-level. Nesting of dependent inductive and non-dependent coinductive types works, however, in full generality.

Before we come to polynomial functors and their fixed points, we show that inductive and coinductive data types actually correspond to initial algebras and final coalgebras, respectively.

Theorem 4.1. *Let $P : \mathbf{E} \to \mathbf{B}$ be a fibration with fibrewise coproducts and dependent sums. If (F,u) with $F : \mathbf{P}_I \to \mathbf{P}_{J_1} \times \cdots \times \mathbf{P}_{J_n}$ is a signature, then there is an isomorphism*

$$\mathrm{DiAlg}\,(F, G_u) \cong \mathrm{Alg}\left(\coprod_{u_1} \circ F_1 +_I \cdots +_I \coprod_{u_n} \circ F_n\right)$$

where $F_k = \pi_k \circ F$ is the kth component of F. In particular, existence of inductive data types and initial algebras coincide. Dually, if P has fibrewise and dependent products, then

$$\mathrm{DiAlg}\,(G_u, F) \cong \mathrm{CoAlg}\left(\prod_{u_1} \circ F_1 \times_I \cdots \times_I \prod_{u_n} \circ F_n\right).$$

In particular, existence of coinductive data types and final coalgebras coincide.

Proof. The first result is given by a simple application of the adjunctions $\coprod_{k=1}^{n} \dashv \Delta_n$ between the (fibrewise) coproduct and the diagonal, and $\coprod_{u_k} \dashv u_k^*$:

$$\frac{\dfrac{FX \longrightarrow G_u X \quad (\text{in } \mathbf{P}_{J_1} \times \cdots \times \mathbf{P}_{J_n})}{(\coprod_{u_1}(F_1 X), \ldots, \coprod_{u_n}(F_n X)) \longrightarrow \Delta_n X \quad (\text{in } \mathbf{P}_I^n)}}{\coprod_{k=1}^{n} \coprod_{u_k}(F_k X) \longrightarrow X \qquad (\text{in } \mathbf{P}_I)}$$

That (di)algebra homomorphisms are preserved follows at once from naturality of the used Hom-set isomorphisms. The correspondence for coinductive types follows by duality. □

To be able to reuse existing work, we work in the following with the codomain fibration $\mathrm{cod} : \mathbf{B}^{\to} \to \mathbf{B}$ for a category $\mathbf{B}$ with pullbacks. Moreover, we assume that $\mathbf{B}$ is locally Cartesian closed, which is equivalent to say that $\mathrm{cod} : \mathbf{B}^{\to} \to \mathbf{B}$ is a closed comprehension category, that is, it is a full CCU with products and coproducts, and $\mathbf{B}$ has a final object, see [17, Thm 10.5.5]. Finally, we need disjoint coproducts in $\mathbf{B}$, which gives us an equivalence $\mathbf{B}/_{I+J} \simeq \mathbf{B}/_I \times \mathbf{B}/_J$, see [17, Prop. 1.5.4].

Definition 4.2. A *dependent polynomial* P indexed by I on variables indexed by J is given by a triple of morphisms

$$J \xleftarrow{\ s\ } B \xrightarrow{\ f\ } A \xrightarrow{\ t\ } I$$

If $J = I = \mathbf{1}$, f is said to be a *(non-dependent) polynomial*. The extension of P is given by the composite

$$[\![P]\!] = \mathbf{B}/_J \xrightarrow{s^*} \mathbf{B}/_B \xrightarrow{\Pi_f} \mathbf{B}/_A \xrightarrow{\coprod_t} \mathbf{B}/_I,$$

which we denote by $[\![f]\!]$ if f is non-dependent. A functor $F : \mathbf{B}/_J \to \mathbf{B}/_I$ is a *dependent polynomial functor*, if there is a dependent polynomial P such that $F \cong [\![P]\!]$.

Remark 4.3. Note that polynomials are called *containers* by Abbott et al. [2, 1], and a polynomial $P = 1 \xleftarrow{!} B \xrightarrow{f} A \xrightarrow{!} 1$ would be written as $A \triangleright f$. Container morphisms, however, are different from those of dependent polynomials, as the latter correspond strong natural transformations [12, Prop. 2.9], whereas the former are in exact correspondence with all natural transformations between extensions [2, Thm. 3.4].

Because of this relation, we will apply results for containers that do not involve morphisms to polynomials. In particular, [2, Prop. 4.1] gives us that we can construct final coalgebras for polynomial functors from initial algebras for polynomial functors. The former are called *M-types* and are denoted by M_f for $f : A \to B$, whereas the latter are *W-types* and denoted by W_f.

Assumption 4.4. We assume that **B** is closed under the formation of W-types, thus is a *Martin-Löf category* in the terminology of [2].

By the above remark, **B** then also has all M-types.

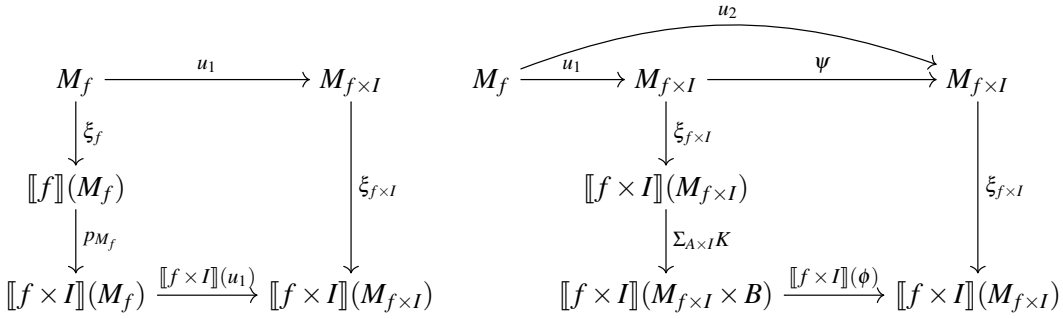
Analogously to how [11, Thm. 12] extends [20, Prop. 3.8], we extend here [6, Thm 3.3]. As it was pointed out by one reviewer, this result is actually in [5], the published version of [6].

Theorem 4.5. *If* **B** *has finite limits, then every dependent polynomial functor has a final coalgebra in* $\mathbf{B}/_I$.

Proof. Let $P = I \xleftarrow{s} B \xrightarrow{f} A \xrightarrow{t} I$ be a dependent polynomial, we construct, analogously to [11] the final coalgebra V of $[\![P]\!]$ as an equaliser as in the following diagram, in which $f \times I$ is a shorthand for $B \times I \xrightarrow{f\times \mathrm{id}_I} A \times I$ and $M_{f\times I}$ is the carrier of the final $[\![f \times I]\!]$-coalgebra.

$$V \xrightarrow{\ g\ } M_f \underset{u_2}{\overset{u_1}{\rightrightarrows}} M_{f\times I}$$

First, we give u_1 and u_2, whose definitions are summarised in the following diagrams.



These diagrams shall indicate that u_1 is given as coinductive extensions and ψ as one-step definition (which can be defined using coproducts), using that $M_{f\times I}$ is a final coalgebra. The maps involved in the diagram are given as follows, which we sometimes spell out in the internal language of cod, see for example [1], as this is sometimes more readable.

- $p : \Sigma_A \Pi_f \Rightarrow \Sigma_{A\times I}\Pi_{f\times I}$ is the natural transformation that maps (a,v) to $(a,t(a),v)$. It is given by the extension $[\![\alpha,\beta]\!] : [\![f]\!] \Rightarrow [\![f\times I]\!]$ of the morphism of polynomials [12]

$$\begin{array}{ccc} B & \xrightarrow{\ f\ } & A \\ {\scriptstyle\beta}\downarrow & \lrcorner & \downarrow{\scriptstyle\alpha} \\ B\times I & \xrightarrow{\ f\times I\ } & A\times I \end{array}$$

  where $\alpha = \langle \mathrm{id}, t\rangle$ and $\beta = \langle \mathrm{id}, t\circ f\rangle$.

- The map $K : \Pi_{f\times I}(M_{f\times I}) \to \Pi_{f\times I}(M_{f\times I}\times B)$ is given as transpose of $\langle \varepsilon_{M_{f\times I}}, \pi_1 \circ \pi\rangle : (f\times I)^*(\Pi_{f\times I}(M_{f\times I})) \to M_{f\times I}\times B$, where ε is the counit of the product (evaluation) and π is the context projection. In the internal language K is given by $K\,v = \lambda(b,i).(v\,(b,i),b)$.

- $\phi : M_{f\times I} \times B \to M_{f\times I}$ is constructed as coinductive extension as in the following diagram

$$\begin{array}{ccc} M_{f\times I}\times B & \xrightarrow{\quad\phi\quad} & M_{f\times I} \\ \downarrow{\scriptstyle \xi_{f\times I}\times \mathrm{id}} & & \\ [\![f\times I]\!](M_{f\times I})\times B & & \downarrow{\scriptstyle \xi_{f\times I}} \\ \downarrow{\scriptstyle e} & & \\ [\![f\times I]\!](M_{f\times I}\times B) & \xrightarrow{[\![f\times I]\!](\phi)} & [\![f\times I]\!](M_{f\times I}) \end{array}$$

  Here e is given by $e((a,i,v),b) = (a, sb, \lambda(b', sb).(v(b',i),b'))$.

The important property, which allows us to prove that $\xi_f : M_f \to [\![f]\!](M_f)$ restricts to $\xi' : V \to [\![P]\!](V)$ and that ξ' is a final coalgebra, is that $x : V_i \iff \xi_f\, x = (a : A, v : \Pi_f M_f), t\, a = i$ and $(\forall b : B. f\, b = a \Rightarrow v\, b : V_{sb})$. The direction from left to right is given by simple a calculation, whereas the other direction can be proved by establishing a bisimulation and between $u_1\, x$ and $u_2\, x$.

Hence V, given as a subobject of M_f, is indeed the final $[\![P]\!]$-coalgebra in $\mathbf{B}/_I$. □

Combining this with [2, Prop. 4.1], we have that the existence of final coalgebras for dependent polynomial functors follows from the existence of initial algebras of (non-dependent) polynomial functors. This gives us the possibility of interpreting non-nested fixed points in any Martin-Löf category as follows.

First, we observe that the equivalence $\mathbf{B}/_{I+J} \simeq \mathbf{B}/_I \times \mathbf{B}/_J$ allows us to rewrite the functors from Thm. 4.1 to a form that is closer to polynomial functors:

$$\coprod_{u_1} \circ F_1 +_I \cdots +_I \coprod_{u_n} \circ F_n \cong \coprod_u F'$$
$$\prod_{u_1} \circ F_1 \times_I \cdots \times_I \prod_{u_n} \circ F_n \cong \prod_u F',$$

where $J = J_1 + \cdots + J_n$, $u : J \to I$ is given by the cotupling $[u_1, \ldots, u_n]$ and $F' : \mathbf{B}/_I \to \mathbf{B}/_J$ is given by $F' = \langle F_1, \ldots, F_n\rangle : \mathbf{B}/_I \to \prod_{i=1}^n \mathbf{B}/_{J_i} \simeq \mathbf{B}/_J$. Thus, if we establish that F' is a polynomial functor, we get that $\coprod_u F'$ and $\prod_u F'$ are polynomial functors, see [1]. For non-nested fixed points, that is, F_k is either a constant functor, given by composition or reindexing, this is immediate, as dependent polynomials can be composed and are closed under constant functors and reindexing, see [12].

We say that a dependent polynomial is *parametric*, if it is of the following form.

$$K + I \xleftarrow{\ s\ } B \xrightarrow{\ f\ } A \xrightarrow{\ t\ } I$$

Such polynomials represent polynomial functors $\mathbf{B}/_K \times \mathbf{B}/_I \to \mathbf{B}/_I$ and allow us speak about nested fixed points just as we have done in Sec. 2. What thus remains is that fixed points of parametric dependent polynomial functors, in the sense of Sec. 2, are again dependent polynomial functors.

The proof of this is literally the same as that for containers [1, Sec. 5.3-5.5] or non-dependent polynomials [11], except that we need to check some extra conditions regarding the indexing.

Theorem 4.6. *Initial algebras and final coalgebras of parametric, dependent polynomial functors are again dependent polynomial functors.*

Proof. Let

$$F = J \xleftarrow{s} B \xrightarrow{f} A \xrightarrow{t} I$$
$$G = I \xleftarrow{u} D \xrightarrow{g} C \xrightarrow{v} I$$

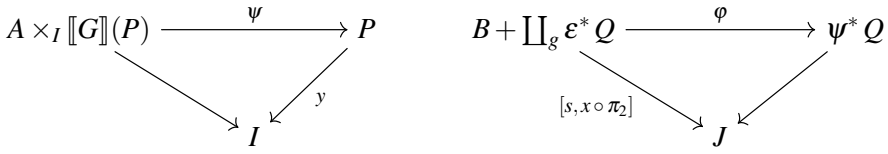
be dependent polynomials and $H(X,Y) = [\![F]\!] \times_I [\![G]\!]$ be the parametric dependent polynomial functor in question. Assuming that there is a polynomial

$$J \xleftarrow{x} Q \xrightarrow{h} P \xrightarrow{y} I$$

so that for $K = \coprod_y \prod_h x^*$ we have $K(X) \cong H(X, K(X))$, we can calculate, as in [1], that we need to have isomorphisms

$$\psi : A \times_I [\![G]\!](P) \cong P$$
$$\varphi : B + \coprod_g \varepsilon^* Q \cong \psi^*(Q)$$

where $B + \coprod_g \varepsilon^* Q$ is, as in loc. cit., is an abbreviation for $B_a + \coprod_{d:D_c} Q(rd)$ in the context $(a,(c,r)) : A \times_I [\![G]\!](P)$. If $K(X)$ shall be an initial algebra, ψ must an initial algebra as well, whereas if $K(X)$ shall be a final coalgebra, ψ must be one. The isomorphism φ is given as the initial $(\psi^{-1})^*(B + \coprod_g \varepsilon^*)$-algebra in both cases, see [1]. This we use to define $x : Q \to J$ as the inductive extension of the map $[s, \pi_2] : (\psi^{-1})^*(B + \coprod_g \varepsilon^* J) \to J$. Given these definitions, the following diagrams commute.



This gives us that the isomorphism given in the proofs of [1, Prop. 5.3.1, Prop. 5.4.2] also work for the dependent polynomial case. The rest of the proofs in loc. cit. go then through, as well. Thus K is in both cases again given by a dependent polynomial. □

Summing up, we are left with the following result.

Corollary 4.7. *All data types for strictly positive signatures can be constructed in any Martin-Löf category.*

Let us see, by means of an example, how the construction in the proof of Thm. 4.5 works intuitively.

Example 4.8. Recall from Ex. 3.4 that partial streams are given by the declaration

codata PStr (A : **Set**) : $\mathbb{N}^\infty \to$ **Set where**
  hd : (n : $\mathbb{N}^\infty$) → PStr (s_∞ n) → A
  tl : (n : $\mathbb{N}^\infty$) → PStr (s_∞ n) → PStr n

By Thm. 4.1, we can construct PStr as the final coalgebra of $F : \mathbf{B}/\mathbf{1} \times \mathbf{B}/\mathbb{N}^\infty \to \mathbf{B}/\mathbb{N}^\infty$ with $F(A,X) = \prod_{s_\infty} !^* A \times \prod_{s_\infty} X$. Note that F is isomorphic to $\mathbf{B}/\mathbf{1} \times \mathbf{B}/\mathbb{N}^\infty \simeq \mathbf{B}/\mathbf{1}+\mathbb{N}^\infty \xrightarrow{[\![P]\!]} \mathbf{B}/\mathbb{N}^\infty$, where P is the polynomial

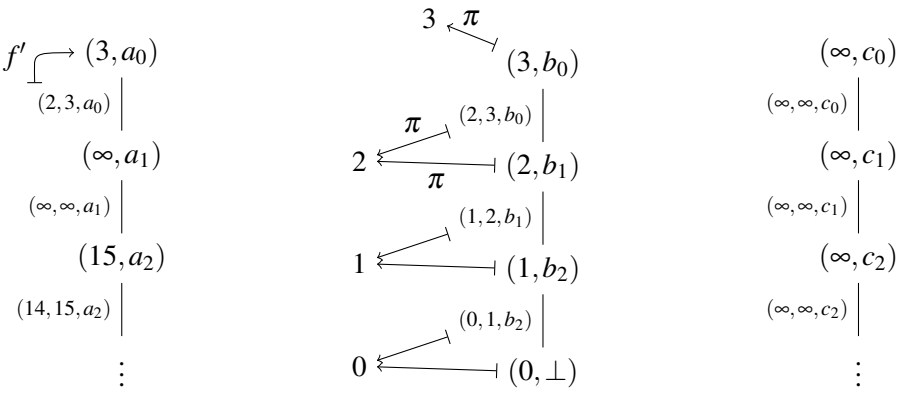
$$P = \mathbf{1} + \mathbb{N}^\infty \xleftarrow{g} 2 \times \mathbb{N}^\infty \xrightarrow{f} \mathbb{N}^\infty \xrightarrow{\mathrm{id}} \mathbb{N}^\infty \qquad g(i,k) = \begin{cases} \kappa_1 *, & i = 1 \\ \kappa_2 k, & i = 2 \end{cases} \qquad f(i,k) = s_\infty k.$$

If we now fix an object $A \in \mathbf{B}/\mathbf{1}$, then $F(A,-) \cong [\![P']\!]$ for the polynomial P' given by

$$P' = \mathbb{N}^\infty \xleftarrow{\pi} \sum_{\mathbb{N}^\infty} \sum_{s_\infty} \prod_{s_\infty} !^* A \xrightarrow{f'} \sum_{\mathbb{N}^\infty} \prod_{s_\infty} !^* A \xrightarrow{\pi} \mathbb{N}^\infty,$$

where π is the projection on the index of a dependent sum and $f'(n,(s_\infty n, v)) = (s_\infty n, v)$.

Recall that we construct in Thm. 4.5 the final coalgebra of $[\![P']\!]$ as a subobject of $M_{f'}$. Below, we present three trees that are elements of $M_{f'}$, where only the second and third are actually selected by the equaliser taken in Thm. 4.5.



Here we denote a pair $(k,v) : \sum_{\mathbb{N}^\infty} \prod_{s_\infty} !^* A$ with $k = s_\infty n$ and $vn = a$ by (k,a), or if $k = 0$ by $(0,\bot)$. Moreover, we indicate the matching of indices in the second tree, which is used to form the equaliser. Note that the second tree is an element of $\mathrm{PStr}(A)\,3$, whereas the third is in $\mathrm{PStr}(A)\,\infty$. □

5 Conclusion and Future Work

We have seen how dependent inductive and coinductive types with type constructors, in the style of Agda, can be given semantics in terms of data type closed categories (DTCC), with the restriction that destructors of coinductive types are not allowed to refer to each other. This situation is summed up in the following table.

Condition	Use/Implications
Cloven fibration	Definition of signatures and data types
Data type completeness	Construction of types indexed by objects in base (e.g., vectors for $\mathbb{N} \in \mathbf{B}$) and types agnostic of indices (e.g., initial and final objects, sums and products)
Data type closedness	Constructed types as index; Full interpretation of data types

Moreover, we have shown that a large part of these data types can be constructed as fixed points of polynomial functors.

Let us finish by discussing directions for future work. First, a full interpretation of syntactic data types has also still to be carried out. Here one has to be careful with type equality, which is usually dealt with using split fibrations and a Beck-Chevalley condition. The latter can be defined generally for the data types of this work, in needs to be checked, however, whether this condition is sufficient for giving a sound interpretation. Finally, the idea of using dialgebras has found its way into the syntax of higher inductive types [7], though in that work the used format of dialgebras is likely to be too liberal to

guarantee the existence of semantics. The reason is that the shape of dialgebras used in the present work ensures that we can construct data types from (co)coalgebras, whereas this is not the case in [7]. Thus it is to be investigated what the right notion of dialgebras is for capturing higher (co)inductive types, such that their semantics in terms of trees can always be constructed.

References

[1] Michael Abbott (2003): *Categories of Containers*. Ph.D. thesis, Leicester.

[2] Michael Abbott, Thorsten Altenkirch & Neil Ghani (2005): *Containers: Constructing strictly positive types*. *Theoretical Computer Science* 342(1), pp. 3–27, doi:10.1016/j.tcs.2005.06.002.

[3] Thorsten Altenkirch & Peter Morris (2009): *Indexed containers*. In: *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, IEEE, pp. 277–285, doi:10.1109/LICS.2009.33.

[4] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg & Anton Setzer (2011): *A Categorical Semantics for Inductive-Inductive Definitions*. In Andrea Corradini, Bartek Klin & Corina Cîrstea, editors: *Algebra and Coalgebra in Computer Science*, *Lecture Notes in Computer Science* 6859, Springer Berlin Heidelberg, pp. 70–84, doi:10.1007/978-3-642-22944-2_6.

[5] Benno van den Berg & Federico De Marchi (2007): *Non-well-founded trees in categories*. *Annals of Pure and Applied Logic* 146(1), pp. 40–59, doi:10.1016/j.apal.2006.12.001.

[6] Benno van den Berg & Federico de Marchi (2004): *Non-well-founded trees in categories*. *arXiv:math/0409158*. Available at `http://arxiv.org/abs/math/0409158`.

[7] Paolo Capriotti (2014): *Mutual and Higher Inductive Types in Homotopy Type Theory*. Available at `http://www.cs.nott.ac.uk/~pvc/away-day-2014/mhit.pdf`.

[8] James Chapman, Pierre-Évariste Dagand, Conor McBride & Peter Morris (2010): *The Gentle Art of Levitation*. In: *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, ACM, New York, NY, USA, pp. 3–14, doi:10.1145/1863543.1863547.

[9] P.-E. Dagand & C. McBride (2013): *A Categorical Treatment of Ornaments*. In: *2013 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS)*, pp. 530–539, doi:10.1109/LICS.2013.60.

[10] Clément Fumex, Neil Ghani & Patricia Johann (2011): *Indexed Induction and Coinduction, Fibrationally*. In Andrea Corradini, Bartek Klin & Corina Cîrstea, editors: *Algebra and Coalgebra in Computer Science*, *LNCS* 6859, Springer, pp. 176–191, doi:10.1007/978-3-642-22944-2_13.

[11] Nicola Gambino & Martin Hyland (2004): *Wellfounded Trees and Dependent Polynomial Functors*. In: *Types for Proofs and Programs*, *LNCS* 3085, Springer, pp. 210–225, doi:10.1007/978-3-540-24849-1_14.

[12] Nicola Gambino & Joachim Kock (2013): *Polynomial functors and polynomial monads*. *Math. Proc. Camb. Philos. Soc.* 154(01), pp. 153–192, doi:10.1017/S0305004112000394. Available at `http://arxiv.org/abs/0906.4931`.

[13] Tatsuya Hagino (1987): *A typed lambda calculus with categorical type constructors*. In: *Category Theory in Computer Science*, pp. 140–157.

[14] Makoto Hamana & Marcelo Fiore (2011): *A Foundation for GADTs and Inductive Families: Dependent Polynomial Functor Approach*. In: *Proceedings of the Seventh Workshop on Generic Programming*, WGP '11, ACM, New York, NY, USA, pp. 59–70, doi:10.1145/2036918.2036927.

[15] Claudio Hermida & Bart Jacobs (1997): *Structural Induction and Coinduction in a Fibrational Setting*. *Inf. Comput.* 145, pp. 107–152, doi:10.1006/inco.1998.2725.

[16] Martin Hofmann (1995): *On the Interpretation of Type Theory in Locally Cartesian Closed Categories*. In: *Proceedings of Computer Science Logic, 8th Workshop, CSL'94, Selected Papers*, *LNCS* 933, Springer, pp. 427–441, doi:10.1007/BFb0022273.

[17] B. Jacobs (1999): *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141, North Holland, Amsterdam.

[18] Jiho Kim (2010): *Higher-order Algebras and Coalgebras from Parameterized Endofunctors*. Electronic Notes in Theoretical Computer Science 264(2), pp. 141–154, doi:10.1016/j.entcs.2010.07.018.

[19] Andres Löh & José Pedro Magalhães (2011): *Generic programming with indexed functors*. In: Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, ACM, pp. 1–12. Available at `http://dl.acm.org/citation.cfm?id=2036920`.

[20] Ieke Moerdijk & Erik Palmgren (2000): *Wellfounded trees in categories*. Annals of Pure and Applied Logic 104(1–3), pp. 189–218, doi:10.1016/S0168-0072(00)00012-9.

[21] R. A. G. Seely (1984): *Locally cartesian closed categories and type theory*. Math. Proc. Camb. Philos. Soc. 95(01), pp. 33–48, doi:10.1017/S0305004100061284.

Equivalence of two Fixed-Point Semantics for Definitional Higher-Order Logic Programs*

Angelos Charalambidis
University of Athens
Athens, Greece
a.charalambidis@di.uoa.gr

Panos Rondogiannis
University of Athens
Athens, Greece
prondo@di.uoa.gr

Ioanna Symeonidou
University of Athens
Athens, Greece
i.symeonidou@di.uoa.gr

Two distinct research approaches have been proposed for assigning a purely extensional semantics to higher-order logic programming. The former approach uses classical domain-theoretic tools while the latter builds on a fixed-point construction defined on a syntactic instantiation of the source program. The relationships between these two approaches had not been investigated until now. In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W. W. Wadge, the two approaches coincide (with respect to ground atoms that involve symbols of the program). On the other hand, we argue that if existential higher-order variables are allowed to appear in the bodies of program rules, the two approaches are in general different. The results of the paper contribute to a better understanding of the semantics of higher-order logic programming.

1 Introduction

Extensional higher-order logic programming has been proposed [10, 1, 2, 7, 5, 4] as a promising generalization of classical logic programming. The key idea behind this paradigm is that the predicates defined in a program essentially denote sets and therefore one can use standard extensional set theory in order to understand their meaning and reason about them. The main difference between the extensional and the more traditional *intensional* approaches to higher-order logic programming [9, 6] is that the latter approaches have a much richer syntax and expressive capabilities but a non-extensional semantics.

Actually, despite the fact that only very few articles have been written regarding extensionality in higher-order logic programming, two main semantic approaches can be identified. The work described in [10, 7, 5, 4] uses classical domain-theoretic tools in order to capture the meaning of higher-order logic programs. On the other hand, the work presented in [1, 2] builds on a fixed-point construction defined on a syntactic instantiation of the source program in order to achieve an extensional semantics. Until now, the relationships between the above two approaches had not yet been investigated.

In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W. W. Wadge [10], the two approaches coincide. Intuitively, this means that for any given definitional program, the sets of true ground atoms of the program are identical under the two different semantic approaches. This result is interesting since it suggests that definitional programs are of fundamental importance for the further study of extensional higher-order logic programming. On the other hand, we argue that if we try to slightly extend the source language, the two approaches give different results in general. Overall, the results of the paper contribute to a better understanding of the

*This research was supported by the project "Handling Uncertainty in Data Intensive Applications", co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Program: THALES, Investing in knowledge society through the European Social Fund.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 18–32, doi:10.4204/EPTCS.191.4

semantics of higher-order logic programming and pave the road for designing a realistic extensional higher-order logic programming language.

The rest of the paper is organized as follows. Section 2 briefly introduces extensional higher-order logic programming and presents in an intuitive way the two existing approaches for assigning meaning to programs of this paradigm. Section 3 contains background material, namely the syntax of definitional programs and the formal details behind the two aforementioned semantic approaches. Section 4 demonstrates the equivalence of the two semantics for definitional programs. Finally, Section 5 concludes the paper with discussion regarding non-definitional programs and with pointers to future work.

2 Intuitive Overview of the two Extensional Approaches

In this section we introduce extensional higher-order logic programming and present the two existing approaches for assigning meaning to programs of this paradigm. Since these two proposals were initially introduced by W. W. Wadge and M. Bezem respectively, we will refer to them as *Wadge's semantics* and *Bezem's semantics* respectively. The key idea behind both approaches is that in order to achieve an extensional semantics, one has to consider a fragment of higher-order logic programming that has a restricted syntax.

2.1 Extensional Higher-Order Logic Programming

The main differences between extensional and intensional higher-order logic programming can be easily understood through two simple examples (borrowed from [5]). Due to space limitations, we avoid a more extensive discussion of this issue; the interested reader can consult [5].

Example 1. Suppose we have a database of professions, both of their membership and their status. We might have rules such as:

```
engineer(tom).
engineer(sally).
programmer(harry).
```

with `engineer` and `programmer` used as predicates. In intensional higher-order logic programming we could also have rules in which these are arguments, eg:

```
profession(engineer).
profession(programmer).
```

Now suppose `tom` and `sally` are also avid users of Twitter. We could have rules:

```
tweeter(tom).
tweeter(sally).
```

The predicates `tweeter` and `engineer` are equal as sets (since they are true for the same objects, namely `tom` and `sally`). If we attempted to understand the above program from an extensional point of view, then we would have to accept that `profession(tweeter)` must also hold (since `tweeter` and `engineer` are indistinguishable as sets). It is clear that the extensional interpretation in this case is completely unnatural. The program can however be understood intensionally: the predicate `profession` is true of the *name* `engineer` (which is different than the name `tweeter`). □

On the other hand, there are cases where predicates can be understood extensionally:

Example 2. Consider a program that consists only of the following rule:

```
p(Q):-Q(0),Q(1).
```

In an extensional language, predicate `p` above can be intuitively understood in purely set-theoretic terms: `p` is the set of all those sets that contain both `0` and `1`.

It should be noted that the above program is also a syntactically acceptable program of the existing intensional logic programming languages. The difference is that in an extensional language the above program has a purely set-theoretic semantics. □

From the above examples it can be understood that extensional higher-order logic programming sacrifices some of the rich syntax of intensional higher-order logic programming in order to achieve semantic clarity.

2.2 Wadge's Semantics

The first proposal for an extensional semantics for higher-order logic programming was given in [10] (and later refined and extended in [7, 5, 4]). The basic idea behind Wadge's approach is that if we consider a properly restricted higher-order logic programming language, then we can use standard ideas from denotational semantics in order to assign an extensional meaning to programs. The basic syntactic assumptions introduced by Wadge in [10] are the following:

- In the head of every rule in a program, each argument of predicate type must be a variable; all such variables must be distinct.
- The only variables of predicate type that can appear in the body of a rule, are variables that appear in its head.

Programs that satisfy the above restrictions are named *definitional* in [10].

Example 3. The program[1]:

```
p(a).
q(b).
r(P,Q):-P(a),Q(b).
```

is definitional because the arguments of predicate type in the head of the rule for `r` are distinct variables. Moreover, the only predicate variables that appear in the body of the same rule, are the variables in its head (namely `P` and `Q`). □

Example 4. The program:

```
q(a).
r(q).
```

is not definitional because the predicate constant `q` appears as an argument in the second clause. For a similar reason, the program in Example 1 is not definitional. The program:

```
p(Q,Q):-Q(a).
```

is also not definitional because the predicate variable `Q` is used twice in the head of the above rule. Finally, the program:

```
p(a):-Q(a).
```

is not definitional because the predicate variable `Q` that appears in the body of the above rule, does not appear in the head of the rule. □

[1] For simplicity reasons, the syntax that we use in our example programs is Prolog-like. The syntax that we adopt in the next section is slightly different and more convenient for the theoretical developments that follow.

As it is argued in [10], if a program satisfies the above two syntactic restrictions, then it has a *unique minimum model* (this notion will be precisely defined in Section 3). Consider again the program of Example 3. In the minimum model of this program, the meaning of predicate `p` is the relation $\{\mathtt{a}\}$ and the meaning of predicate `q` is the relation $\{\mathtt{b}\}$. On the other hand, the meaning of predicate `r` in the minimum model is a relation that contains the pairs $(\{\mathtt{a}\},\{\mathtt{b}\})$, $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{b}\})$, $(\{\mathtt{a}\},\{\mathtt{a},\mathtt{b}\})$ and $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{a},\mathtt{b}\})$. As remarked by W. W. Wadge (and formally demonstrated in [7, 5]), the minimum model of every definitional program is monotonic and continuous[2]. Intuitively, monotonicity means that if in the minimum model the meaning of a predicate is true of a relation, then it is also true of every superset of this relation. For example, we see that since the meaning of `r` is true of $(\{\mathtt{a}\},\{\mathtt{b}\})$, then it is also true of $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{b}\})$ (because $\{\mathtt{a},\mathtt{b}\}$ is a superset of $\{\mathtt{a}\}$).

The minimum model of a given definitional program can be constructed as the least fixed-point of an operator that is associated with the program, called the *immediate consequence operator* of the program. As it is demonstrated in [10, 7], the immediate consequence operator is monotonic, and this guarantees the existence of the least fixed-point which is constructed by a bottom-up iterative procedure (more formal details will be given in the next section).

Example 5. Consider the definitional program:

```
q(a).
q(b).
p(Q):-Q(a).
id(R)(X):-R(X).
```

In the minimum model of the above program, the meaning of `q` is the relation $\{\mathtt{a},\mathtt{b}\}$. The meaning of `p` is the set of all relations that contain (at least) `a`; more formally, it is the relation $\{r \mid \mathtt{a} \in r\}$. The meaning of `id` is the set of all pairs (r,d) such that d belongs to r; more formally, it is the relation $\{(r,d) \mid d \in r\}$. □

Notice that in the construction of the minimum model, all predicates are initially assigned the empty relation. The rules of the program are then used in order to improve the meaning assigned to each predicate symbol. More specifically, at each step of the fixed-point computation, the meaning of each predicate symbol either stabilizes or it becomes richer than the previous step.

Example 6. Consider again the definitional program of the previous example. In the iterative construction of the minimum model, all predicates are initially assigned the empty relation (of the corresponding type). After the first step of the construction, the meaning assigned to predicate `q` is the relation $\{\mathtt{a},\mathtt{b}\}$ due to the first two facts of the program. At this same step, the meaning of `p` becomes the relation $\{r \mid \mathtt{a} \in r\}$. Also, the meaning of `id` becomes equal to the relation $\{(r,d) \mid d \in r\}$. Additional iterations will not alter the relations we have obtained at the first step; in other words, we have reached the fixed-point of the bottom-up computation. □

In the above example, we obtained the meaning of the program in just one step. If the source program contained recursive definitions, convergence to the least fixed-point would in general require more steps.

2.3 Bezem's Semantics

In [1, 2], M. Bezem proposed an alternative extensional semantics for higher-order logic programs. Again, the syntax of the source language has to be appropriately restricted. Actually, the class of programs adopted in [1, 2] is a proper superset of the class of definitional programs. In particular, Bezem proposes the class of *hoapata programs* which extend definitional programs:

[2] The notion of continuity will not play any role in the remaining part of this paper.

- A predicate variable that appears in the body of a rule, need not necessarily appear in the head of that rule.
- The head of a rule can be an atom that starts with a predicate variable.

Example 7. All definitional programs of the previous subsection are also hoapata. The following non-definitional program of Example 4 is hoapata:

```
p(a):-Q(a).
```

Intuitively, the above program states that `p` is true of `a` if there exists a predicate that is defined in the program that is true of `a`. We will use this program in our discussion at the end of the paper.

The following program is also hoapata (but not definitional):

```
P(a,b).
```

Intuitively, the above program states that every binary relation is true of the pair (`a`,`b`). □

Given a hoapata program, the starting idea behind Bezem's approach is to take its "ground instantiation" in which we replace variables with well-typed terms of the Herbrand Universe of the program (ie., terms that can be created using only predicate and individual constants that appear in the program). For example, given the program:

```
q(a).
q(b).
p(Q):-Q(a).
id(R)(X):-R(X).
```

the ground instantiation is the following infinite "program":

```
q(a).
q(b).
p(q):-q(a).
id(q)(a):-q(a).
p(id(q)):-id(q)(a).
id(id(q))(a):-id(q)(a).
p(id(id(q))):-id(id(q))(a).
          ...
```

One can now treat the new program as an infinite propositional one (ie., each ground atom can be seen as a propositional one). This implies that we can use the standard least fixed-point construction of classical logic programming (see for example [8]) in order to compute the set of atoms that should be taken as "true". In our example, the least fixed-point will contain atoms such as `q(a)`, `q(b)`, `p(q)`, `id(q)(a)`, `p(id(q))`, and so on.

A main contribution of Bezem's work was that he established that the least fixed-point of the ground instantiation of every hoapata program is *extensional*. This notion can intuitively be explained as follows. It is obvious in the above example that the relations `q` and `id(q)` are equal (they are both true of only the constant `a`, and therefore they both correspond to the relation {`a`}). Therefore, we would expect that (for example) if `p(q)` is true then `p(id(q))` is also true because `q` and `id(q)` should be considered as interchangeable. This property of "interchangeability" is formally defined in [1, 2] and it is demonstrated that it holds in the least fixed-point of the immediate consequence operator of the ground instance of every hoapata program.

2.4 The Differences Between the two Approaches

It is not hard to see that the two semantic approaches outlined in the previous subsections, have some important differences. First, they operate on different source languages. Therefore, in order to compare them we have to restrict Bezem's approach to the class of definitional programs[3].

The main difference however between the two approaches is the way that the least fixed-point of the immediate consequence operator is constructed in each case. In Wadge's semantics the construction starts by initially assigning to every predicate constant the empty relation; these relations are then improved at each step until they converge to their final meaning. In other words, Wadge's semantics *manipulates relations*. On the other hand, Bezem's semantics works with the ground instantiation of the source program and, at first sight, it appears to have a more syntactic flavor. In our running example, Wadge's approach converges in a single step while Bezem's approach takes an infinite number of steps in order to converge. However, one can easily verify that the ground atoms that belong to the least fixed-point under Bezem's semantics, are also true in the minimum model under Wadge's semantics. This poses the question whether under both approaches, the sets of ground atoms that are true, are identical. This is the question that we answer positively in the rest of the paper.

3 Definitional Programs and their Semantics

In this section we define the source language $\mathscr{H}$ of definitional higher-order logic programs. Moreover, we present in a formal way the two different extensional semantics that have been proposed for such programs, namely Wadge's and Bezem's semantics respectively.

3.1 Syntax

The language $\mathscr{H}$ is based on a simple type system that supports two base types: o, the boolean domain, and ι, the domain of individuals (data objects). The composite types are partitioned into three classes: functional (assigned to function symbols), predicate (assigned to predicate symbols) and argument (assigned to parameters of predicates).

Definition 1. A type can either be functional, argument, or predicate, denoted by σ, ρ and π respectively and defined as:

$$\sigma := \iota \mid (\iota \to \sigma)$$
$$\pi := o \mid (\rho \to \pi)$$
$$\rho := \iota \mid \pi$$

We will use τ to denote an arbitrary type (either functional, argument or predicate one). As usual, the binary operator $\to$ is right-associative. A functional type that is different than ι will often be written in the form $\iota^n \to \iota$, $n \geq 1$. Moreover, it can be easily seen that every predicate type π can be written in the form $\rho_1 \to \cdots \to \rho_n \to o$, $n \geq 0$ (for $n = 0$ we assume that $\pi = o$).

We proceed by defining the syntax of $\mathscr{H}$:

Definition 2. The alphabet of the higher-order language $\mathscr{H}$ consists of the following:

1. Predicate variables of every predicate type π (denoted by capital letters such as $\mathsf{P}, \mathsf{Q}, \mathsf{R}, \ldots$).

[3] Actually, we could alternatively extend Wadge's approach to a broader class of programs. Such an extension has already been performed in [5], and we will discuss its repercussions in the concluding section.

2. Individual variables of type ι (denoted by capital letters such as $\mathsf{X},\mathsf{Y},\mathsf{Z},\ldots$).
3. Predicate constants of every predicate type π (denoted by lowercase letters such as $\mathsf{p},\mathsf{q},\mathsf{r},\ldots$).
4. Individual constants of type ι (denoted by lowercase letters such as $\mathsf{a},\mathsf{b},\mathsf{c},\ldots$).
5. Function symbols of every functional type $\sigma \neq \iota$ (denoted by lowercase letters such as $\mathsf{f},\mathsf{g},\mathsf{h},\ldots$).
6. The logical conjunction constant $\wedge$, the inverse implication constant $\leftarrow$, the left and right parentheses, and the equality constant $\approx$ for comparing terms of type ι.

The set consisting of the predicate variables and the individual variables of $\mathscr{H}$ will be called the set of *argument variables* of $\mathscr{H}$. Argument variables will be usually denoted by V and its subscripted versions.

Definition 3. The set of *terms* of the higher-order language $\mathscr{H}$ is defined as follows:

- Every predicate variable (respectively predicate constant) of type π is a term of type π; every individual variable (respectively individual constant) of type ι is a term of type ι;
- if f is an n-ary function symbol and $\mathsf{E}_1,\ldots,\mathsf{E}_n$ are terms of type ι then $(\mathsf{f}\ \mathsf{E}_1\cdots\mathsf{E}_n)$ is a term of type ι;
- if E_1 is a term of type $\rho\rightarrow\pi$ and E_2 a term of type ρ then $(\mathsf{E}_1\ \mathsf{E}_2)$ is a term of type π.

Definition 4. The set of *expressions* of the higher-order language $\mathscr{H}$ is defined as follows:

- A term of type ρ is an expression of type ρ;
- if E_1 and E_2 are terms of type ι, then $(\mathsf{E}_1\approx\mathsf{E}_2)$ is an expression of type o.

We write $vars(\mathsf{E})$ to denote the set of all the variables in E. Expressions (respectively terms) that have no variables will often be referred to as *ground expressions* (respectively *ground terms*). Expressions of type o will often be referred to as *atoms*. We will omit parentheses when no confusion arises. To denote that an expression E has type ρ we will often write $\mathsf{E}:\rho$.

Definition 5. A *clause* is a formula $\mathsf{p}\,\mathsf{V}_1\cdots\mathsf{V}_n\leftarrow\mathsf{E}_1\wedge\cdots\wedge\mathsf{E}_m$, where p is a predicate constant, $\mathsf{p}\,\mathsf{V}_1\cdots\mathsf{V}_n$ is a term of type o and $\mathsf{E}_1,\ldots,\mathsf{E}_m$ are expressions of type o. The term $\mathsf{p}\,\mathsf{V}_1\cdots\mathsf{V}_n$ is called the *head* of the clause, the variables $\mathsf{V}_1,\ldots,\mathsf{V}_n$ are the *formal parameters* of the clause and the conjunction $\mathsf{E}_1\wedge\cdots\wedge\mathsf{E}_m$ is its *body*. A *definitional clause* is a clause that additionally satisfies the following two restrictions:

1. All the formal parameters are distinct variables (ie., for all i,j such that $1\leq i,j\leq n$, $\mathsf{V}_i\neq\mathsf{V}_j$).
2. The only variables that can appear in the body of the clause are its formal parameters and possibly some additional individual variables (namely variables of type ι).

A *program* P is a set of definitional program clauses.

In the rest of the paper, when we refer to "clauses" we will mean definitional ones. For simplicity, we will follow the usual logic programming convention and we will write $\mathsf{p}\,\mathsf{V}_1\cdots\mathsf{V}_n\leftarrow\mathsf{E}_1,\ldots,\mathsf{E}_m$ instead of $\mathsf{p}\,\mathsf{V}_1\cdots\mathsf{V}_n\leftarrow\mathsf{E}_1\wedge\cdots\wedge\mathsf{E}_m$.

Our syntax differs slightly from the Prolog-like syntax that we have used in Section 2. However, one can easily verify that we can transform every program from the former syntax to the latter.

Definition 6. For a program P, we define the Herbrand universe for every argument type ρ, denoted by $U_{\mathsf{P},\rho}$ to be the set of all ground terms of type ρ, that can be formed out of the individual constants, function symbols and predicate constants in the program.

In the following, we will often talk about the "ground instantiation of a program". This notion is formally defined below.

Definition 7. A *ground substitution* θ is a finite set of the form $\{\mathsf{V}_1/\mathsf{E}_1,\ldots,\mathsf{V}_n/\mathsf{E}_n\}$ where the V_i's are different argument variables and each E_i is a ground term having the same type as V_i. We write $dom(\theta)=\{\mathsf{V}_1,\ldots,\mathsf{V}_n\}$ to denote the domain of θ.

We can now define the application of a substitution to an expression.

Definition 8. Let θ be a substitution and E be an expression. Then, $\mathsf{E}\theta$ is an expression obtained from E as follows:

- $\mathsf{E}\theta=\mathsf{E}$ if E is a predicate or individual constant;
- $\mathsf{V}\theta=\theta(\mathsf{V})$ if $\mathsf{V}\in dom(\theta)$; otherwise, $\mathsf{V}\theta=\mathsf{V}$;
- $(\mathsf{f}\ \mathsf{E}_1\cdots\mathsf{E}_n)\theta=(\mathsf{f}\ \mathsf{E}_1\theta\cdots\mathsf{E}_n\theta)$;
- $(\mathsf{E}_1\ \mathsf{E}_2)\theta=(\mathsf{E}_1\theta\ \mathsf{E}_2\theta)$;
- $(\mathsf{E}_1\approx\mathsf{E}_2)\theta=(\mathsf{E}_1\theta\approx\mathsf{E}_2\theta)$.

Definition 9. Let E be an expression and θ be a ground substitution such that $vars(\mathsf{E})\subseteq dom(\theta)$. Then, the ground expression $\mathsf{E}\theta$ is called a *ground instantiation* of E. A *ground instantiation of a clause* $\mathsf{p}\ \mathsf{V}_1\cdots\mathsf{V}_n\leftarrow\mathsf{E}_1,\ldots,\mathsf{E}_m$ with respect to a ground substitution θ is the formula $(\mathsf{p}\ \mathsf{V}_1\cdots\mathsf{V}_n)\theta\leftarrow\mathsf{E}_1\theta,\ldots,\mathsf{E}_m\theta$. The *ground instantiation of a program* P is the (possibly infinite) set that contains all the ground instantiations of the clauses of P with respect to all possible ground substitutions.

3.2 Wadge's Semantics

The key idea behind Wadge's semantics is (intuitively) to assign to program predicates monotonic relations. In the following, given posets A and B, we write $[A\stackrel{m}{\rightarrow}B]$ to denote the set of all monotonic relations from A to B.

Before specifying the semantics of expressions of $\mathscr{H}$ we need to provide the set-theoretic meaning of the types of expressions of $\mathscr{H}$ with respect to an underlying domain. It is customary in logic programming to take the underlying domain to be the Herbrand universe $U_{\mathsf{P},\iota}$. In the following definition we define simultaneously and recursively two things: the semantics $[\![\tau]\!]$ of a type τ and a corresponding partial order $\sqsubseteq_\tau$ on the elements of $[\![\tau]\!]$. We adopt the usual ordering of the truth values *false* and *true*, i.e. *false* $\leq$ *false*, *true* $\leq$ *true* and *false* $\leq$ *true*.

Definition 10. Let P be a program. Then,

- $[\![\iota]\!]=U_{\mathsf{P},\iota}$ and $\sqsubseteq_\iota$ is the trivial partial order that relates every element to itself;
- $[\![\iota^n\rightarrow\iota]\!]=U^n_{\mathsf{P},\iota}\rightarrow U_{\mathsf{P},\iota}$. A partial order for this case is not needed;
- $[\![o]\!]=\{\mathit{false},\mathit{true}\}$ and $\sqsubseteq_o$ is the partial order $\leq$ on truth values;
- $[\![\rho\rightarrow\pi]\!]=[[\![\rho]\!]\stackrel{m}{\rightarrow}[\![\pi]\!]]$ and $\sqsubseteq_{\rho\rightarrow\pi}$ is the partial order defined as follows: for all $f,g\in[\![\rho\rightarrow\pi]\!]$, $f\sqsubseteq_{\rho\rightarrow\pi}g$ iff $f(d)\sqsubseteq_\pi g(d)$ for all $d\in[\![\rho]\!]$.

We now proceed to define Herbrand interpretations and states.

Definition 11. A Herbrand interpretation I of a program P is an interpretation such that:

1. for every individual constant c that appears in P, $I(\mathsf{c})=\mathsf{c}$;
2. for every predicate constant $\mathsf{p}:\pi$ that appears in P, $I(\mathsf{p})\in[\![\pi]\!]$;
3. for every n-ary function symbol f that appears in P and for all $\mathsf{t}_1,\ldots\mathsf{t}_n\in U_{\mathsf{P},\iota}$, $I(\mathsf{f})\ \mathsf{t}_1\ \cdots\mathsf{t}_n=\mathsf{f}\ \mathsf{t}_1\ \cdots\mathsf{t}_n$.

Definition 12. A Herbrand state s of a program P is a function that assigns to each argument variable V of type ρ, an element $s(\mathsf{V}) \in [\![\rho]\!]$.

In the following, $s[\mathsf{V}/d]$ is used to denote a state that is identical to s the only difference being that the new state assigns to V the value d.

Definition 13. Let P be a program, I be a Herbrand interpretation of P and s be a Herbrand state. Then, the semantics of the expressions of P is defined as follows:

1. $[\![\mathsf{V}]\!]_s(I) = s(\mathsf{V})$ if V is a variable;
2. $[\![\mathsf{c}]\!]_s(I) = I(\mathsf{c})$ if c is an individual constant;
3. $[\![\mathsf{p}]\!]_s(I) = I(\mathsf{p})$ if p is a predicate constant;
4. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I) \cdots [\![\mathsf{E}_n]\!]_s(I)$;
5. $[\![(\mathsf{E}_1\ \mathsf{E}_2)]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\ [\![\mathsf{E}_2]\!]_s(I)$;
6. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(I) = \mathit{true}$ if $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$ and *false* otherwise.

For ground expressions E we will often write $[\![\mathsf{E}]\!](I)$ instead of $[\![\mathsf{E}]\!]_s(I)$ since the meaning of E is independent of s.

It is straightforward to confirm that the above definition assigns to every expression an element of the corresponding semantic domain, as stated in the following lemma:

Lemma 1. *Let P be a program and let $\mathsf{E} : \rho$ be an expression. Also, let I be a Herbrand interpretation and s be a Herbrand state. Then $[\![\mathsf{E}]\!]_s(I) \in [\![\rho]\!]$.*

Definition 14. Let P be a program and M be a Herbrand interpretation of P. Then, M is a Herbrand model of P iff for every clause $\mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1, \ldots \mathsf{E}_m$ in P and for every Herbrand state s, if for all $i \in \{1, \ldots, m\}$, $[\![\mathsf{E}_i]\!]_s(M) = \mathit{true}$ then $[\![\mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = \mathit{true}$.

In the following we denote the set of Herbrand interpretations of a program P with $\mathscr{I}_\mathsf{P}$. We define a partial order on $\mathscr{I}_\mathsf{P}$ as follows: for all $I, J \in \mathscr{I}_\mathsf{P}$, $I \sqsubseteq_{\mathscr{I}_\mathsf{P}} J$ iff for every predicate $\mathsf{p} : \pi$ that appears in P, $I(\mathsf{p}) \sqsubseteq_\pi J(\mathsf{p})$. Similarly, we denote the set of Herbrand states with $\mathscr{S}_\mathsf{P}$ and we define a partial order as follows: for all $s_1, s_2 \in \mathscr{S}_\mathsf{P}$, $s_1 \sqsubseteq_{\mathscr{S}_\mathsf{P}} s_2$ iff for all variables $\mathsf{V} : \rho$, $s_1(\mathsf{V}) \sqsubseteq_\rho s_2(\mathsf{V})$. The following lemmata are straightforward to establish:

Lemma 2. *Let P be a program. Then, $(\mathscr{I}_\mathsf{P}, \sqsubseteq_{\mathscr{I}_\mathsf{P}})$ is a complete lattice.*

Lemma 3. *Let P be a program and let $\mathsf{E} : \rho$ be an expression. Let I, J be Herbrand interpretations and s, s' be Herbrand states. Then,*

1. *If $I \sqsubseteq_{\mathscr{I}_\mathsf{P}} J$ then $[\![\mathsf{E}]\!]_s(I) \sqsubseteq_\rho [\![\mathsf{E}]\!]_s(J)$.*
2. *If $s \sqsubseteq_{\mathscr{I}_\mathsf{P}} s'$ then $[\![\mathsf{E}]\!]_s(I) \sqsubseteq_\rho [\![\mathsf{E}]\!]_{s'}(I)$.*

We can now define the *immediate consequence operator* for $\mathscr{H}$ programs, which generalizes the corresponding operator for classical (first-order) programs [8].

Definition 15. Let P be a program. The mapping $T_\mathsf{P} : \mathscr{I}_\mathsf{P} \to \mathscr{I}_\mathsf{P}$ is called the *immediate consequence operator for* P and is defined for every predicate $\mathsf{p} : \rho_1 \to \cdots \to \rho_n \to o$ and $d_i \in [\![\rho_i]\!]$ as

$$T_\mathsf{P}(I)(\mathsf{p})\ d_1 \cdots d_n = \begin{cases} \mathit{true} & \text{there exists a clause } \mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1, \ldots \mathsf{E}_m \text{ such that} \\ & \text{for every state } s,\ [\![\mathsf{E}_i]\!]_{s[\mathsf{V}_1/d_1, \ldots, \mathsf{V}_n/d_n]}(I) = \mathit{true} \text{ for all } i \in \{1, \ldots, m\} \\ \mathit{false} & \text{otherwise.} \end{cases}$$

It is not hard to see that T_P is a monotonic function, and this leads to the following theorem [10, 7]:

Theorem 1. *Let P be a program. Then $M_\mathsf{P} = \mathit{lfp}(T_\mathsf{P})$ is the minimum, with respect to $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$, Herbrand model of P.*

3.3 Bezem's Semantics

In contrast to Wadge's semantics which proceeds by constructing the meaning of predicates as relations, Bezem's approach takes a (seemingly) more syntax-oriented approach. In particular, Bezem's approach builds on the ground instantiation of the source program in order to retrieve the meaning of the program. In our definitions below, we follow relatively closely the exposition given in [1, 2, 3].

Definition 16. Let P be a program and let $\mathsf{Gr}(\mathsf{P})$ be its ground instantiation. An interpretation I for $\mathsf{Gr}(\mathsf{P})$ is defined as a subset of $U_{\mathsf{P},o}$ by the usual convention that, for any $\mathsf{A} \in U_{\mathsf{P},o}$, $I(\mathsf{A}) = \mathit{true}$ iff $\mathsf{A} \in I$. We also extend the interpretation I for every $(\mathsf{E}_1 \approx \mathsf{E}_2)$ atom as follows: $I(\mathsf{E}_1 \approx \mathsf{E}_2) = \mathit{true}$ if $\mathsf{E}_1 = \mathsf{E}_2$ and *false* otherwise.

Observe that the meaning of $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is fixed and independent of the interpretation.

Definition 17. We define the immediate consequence operator, $\mathscr{T}_{\mathsf{Gr}(\mathsf{P})}$, of P as follows:

$$\mathscr{T}_{\mathsf{Gr}(\mathsf{P})}(I)(\mathsf{A}) = \begin{cases} \mathit{true} & \text{if there exists a clause } \mathsf{A} \leftarrow \mathsf{E}_1, \ldots, \mathsf{E}_m \text{ in } \mathsf{Gr}(\mathsf{P}) \\ & \text{such that } I(\mathsf{E}_i) = \mathit{true} \text{ for all } i \in \{1, \ldots, m\} \\ \mathit{false} & \text{otherwise.} \end{cases}$$

As it is well established in bibliography (for example [8]), the least fixed-point of the immediate consequence operator of a propositional program exists and is the minimum, with respect to set inclusion and equivalently $\leq$, model of $\mathsf{Gr}(\mathsf{P})$. This fixed-point, which we will henceforth denote by $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, is shown in [1, 2] to be directly related to a notion of a model capable of capturing the perceived semantics of the higher-order program P. In particular, this model by definition assigns to all ground atoms the same truth values as $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$. It is therefore justified that we restrict our attention to $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, instead of the aforementioned higher-order model, in our attempt to prove the equivalence of Bezem's semantics and Wadge's semantics.

The following definition and subsequent theorem obtained in [3], identify a property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ that we will need in the next section.

Definition 18. Let P be a program and let $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. For every argument type ρ we define a corresponding partial order as follows: for type ι, we define $\preceq_\iota$ as syntactical equality, i.e. $\mathsf{E} \preceq_\iota \mathsf{E}$ for all $\mathsf{E} \in U_{\mathsf{P},\iota}$. For type o, $\mathsf{E} \preceq_o \mathsf{E}'$ iff $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}) \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}')$. For a predicate type of the form $\rho \to \pi$, $\mathsf{E} \preceq_{\rho\to\pi} \mathsf{E}'$ iff $\mathsf{E}\,\mathsf{D} \preceq_\pi \mathsf{E}'\,\mathsf{D}$ for all $\mathsf{D} \in U_{\mathsf{P},\rho}$.

Theorem 2 ($\preceq$-Monotonicity Property)**.** *[3] Let P be a program and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. Then for all $\mathsf{E} \in U_{\mathsf{P},\rho\to\pi}$ and all $\mathsf{D}, \mathsf{D}' \in U_{\mathsf{P},\rho}$ such that $\mathsf{D} \preceq_\rho \mathsf{D}'$, it holds $\mathsf{E}\,\mathsf{D} \preceq_\pi \mathsf{E}\,\mathsf{D}'$.*

4 Equivalence of the two Semantics

In this section we demonstrate that the two semantics presented in the previous section, are equivalent for definitional programs. To help us transcend the differences between these approaches, we introduce two key notions, namely that of the *ground restriction* of a higher-order interpretation and its complementary notion of the *semantic extension* of ground expressions. But first we present the following *Substitution Lemma*, which will be useful in the proofs of later results.

Lemma 4 (Substitution Lemma)**.** *Let P be a program and I be a Herbrand interpretation of P. Also let E be an expression and θ be a ground substitution with $vars(\mathsf{E}) \subseteq dom(\theta)$. If s is a Herbrand state such that, for all $\mathsf{V} \in vars(\mathsf{E})$, $s(\mathsf{V}) = [\![\theta(\mathsf{V})]\!](I)$, then $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$.*

Proof. By a structural induction on E. For the basis case, if $\mathsf{E} = \mathsf{p}$ or $\mathsf{E} = \mathsf{c}$ then the statement reduces to an identity and if $\mathsf{E} = \mathsf{V}$ then it holds by assumption. For the induction step, we first examine the case that $\mathsf{E} = (\mathsf{f}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$; then $[\![\mathsf{E}]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I)\ \cdots\ [\![\mathsf{E}_n]\!]_s(I)$ and $[\![\mathsf{E}\theta]\!](I) = I(\mathsf{f})\ [\![\mathsf{E}_1\theta]\!](I)\ \cdots\ [\![\mathsf{E}_n\theta]\!](I)$. By the induction hypothesis, $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_1\theta]\!](I), \ldots, [\![\mathsf{E}_n]\!]_s(I) = [\![\mathsf{E}_n\theta]\!](I)$, thus we have $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$. Now consider the case that $\mathsf{E} = \mathsf{E}_1\,\mathsf{E}_2$. We have $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)[\![\mathsf{E}_2]\!]_s(I)$ and $[\![\mathsf{E}\theta]\!](I) = [\![\mathsf{E}_1\theta]\!](I)[\![\mathsf{E}_2\theta]\!](I)$. Again, applying the induction hypothesis, we conclude that $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$. Finally, if $\mathsf{E} = (\mathsf{E}_1 \approx \mathsf{E}_2)$ we have that $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$, which, by the induction hypothesis, holds iff $[\![\mathsf{E}_1\theta]\!](I) = [\![\mathsf{E}_2\theta]\!](I)$. Moreover, we have $[\![\mathsf{E}\theta]\!](I) = \mathit{true}$ iff $[\![\mathsf{E}_1\theta]\!](I) = [\![\mathsf{E}_2\theta]\!](I)$, therefore we conclude that $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $[\![\mathsf{E}\theta]\!](I) = \mathit{true}$. □

Given a Herbrand interpretation I of a definitional program, it is straightforward to devise a corresponding interpretation of the ground instantiation of the program, by restricting I to only assigning truth values to ground atoms. As expected, such a restriction of a model of the program produces a model of its ground instantiation. This idea is formalized in the following definition and theorem.

Definition 19. Let P be a program, I be a Herbrand interpretation of P and $\mathsf{Gr}(\mathsf{P})$ be the ground instantiation of P. We define the *ground restriction* of I, which we denote by $I|_{\mathsf{Gr}(\mathsf{P})}$, to be an interpretation of $\mathsf{Gr}(\mathsf{P})$, such that, for every ground atom A, $I|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{A}) = [\![\mathsf{A}]\!](I)$.

Theorem 3. *Let* P *be a program and* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation. Also let* M *be a Herbrand model of* P *and* $M|_{\mathsf{Gr}(\mathsf{P})}$ *be the ground restriction of* M*. Then* $M|_{\mathsf{Gr}(\mathsf{P})}$ *is a model of* $\mathsf{Gr}(\mathsf{P})$*.*

Proof. By definition, each clause in $\mathsf{Gr}(\mathsf{P})$ is of the form $\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n \leftarrow \mathsf{B}_1\theta, \ldots, \mathsf{B}_k\theta$, i.e. the ground instantiation of a clause $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_\mathsf{n} \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_k$ in P with respect to a ground substitution θ, such that $\mathit{dom}(\theta)$ includes $V_1, \ldots, V_n$ and all other (individual) variables appearing in the body of the clause and $\theta(\mathsf{V}_i) = \mathsf{E}_i$, for all $i \in \{1, \ldots, n\}$. Let s be a Herbrand state such that $s(\mathsf{V}) = [\![\theta(\mathsf{V})]\!](M)$, for all $V \in \mathit{dom}(\theta)$. By the Substitution Lemma (Lemma 4) and the definition of $M|_{\mathsf{Gr}(\mathsf{P})}$, $[\![\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = [\![\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n]\!](M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n)$. Similarly, for each atom B_i in the body of the clause, we have $[\![\mathsf{B}_i]\!]_s(M) = [\![\mathsf{B}_i\theta]\!](M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{B}_i\theta), 1 \leq i \leq k$. Consequently, if $M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{B}_i\theta) = \mathit{true}$ for all $i \in \{1, \ldots, k\}$, we also have that $[\![\mathsf{B}_i]\!]_s(M) = \mathit{true}, 1 \leq i \leq k$. As M is a model of P, this implies that $[\![\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{true}$ and therefore $M|_{\mathsf{Gr}(\mathsf{P})}$ is a model of $\mathsf{Gr}(\mathsf{P})$. □

The above theorem is of course useful in connecting the $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$-minimum Herbrand model of a program to its ground instantiation. However, in order to prove the equivalence of the two semantics under consideration, we will also need to go in the opposite direction and connect the $\leq$-minimum model of the ground program to the higher-order program. To this end we introduce the previously mentioned *semantic extensions* of a ground expression.

Definition 20. Let P be a program and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. Let E be a ground expression of argument type ρ and d be an element of $[\![\rho]\!]$. We will say that d is a semantic extension of E and write $d \rhd_\rho \mathsf{E}$ if

- $\rho = \iota$ and $d = \mathsf{E}$;
- $\rho = o$ and $d = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E})$;
- $\rho = \rho' \to \pi$ and for all $d' \in [\![\rho']\!]$ and $\mathsf{E}' \in U_{P,\rho'}$, such that $d' \rhd_{\rho'} \mathsf{E}'$, it holds that $d\,d' \rhd_\pi \mathsf{E}\,\mathsf{E}'$.

Compared to that of the ground restriction presented earlier, the notion of extending a syntactic object to the realm of semantic elements, is more complicated. In fact, even the existence of a semantic extension is not immediately obvious. The next lemma guarantees that not only can such an extension

be constructed for any expression of the language, but it also has an interesting property of mirroring the ordering of semantic objects with respect to $\sqsubseteq_\tau$ in a corresponding ordering of the expressions with respect to $\preceq_\tau$.

Lemma 5. *Let* P *be a program,* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation and* $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ *be the* $\leq$*-minimum model of* $\mathsf{Gr}(\mathsf{P})$*. For every argument type* ρ *and every ground term* $\mathsf{E} \in U_{\mathsf{P},\rho}$

1. *There exists* $e \in [\![\rho]\!]$ *such that* $e \rhd_\rho \mathsf{E}$.
2. *For all* $e, e' \in [\![\rho]\!]$ *and all* $\mathsf{E}' \in U_{\mathsf{P},\rho}$*, if* $e \rhd_\rho \mathsf{E}$*,* $e' \rhd_\rho \mathsf{E}'$ *and* $e \sqsubseteq_\rho e'$*, then* $\mathsf{E} \preceq_\rho \mathsf{E}'$.

Proof. We prove both statements simultaneously, performing an induction on the structure of ρ. Specifically, the first statement is proven by showing that in each case we can construct a function e of type ρ, which is monotonic with respect to $\sqsubseteq_\rho$ and satisfies $e \rhd_\rho \mathsf{E}$.

In the basis case, the construction of e for types ι and o is trivial. Also, if $\rho = \iota$, then both $\rhd_\rho$ and $\sqsubseteq_\rho$ reduce to equality, so we have $\mathsf{E} = \mathsf{E}'$, which in this case is equivalent to $\mathsf{E} \preceq_\rho \mathsf{E}'$. On the other hand, for $\rho = o$, $\rhd_\rho$ identifies with equality, while $\sqsubseteq_\rho$ and $\preceq_\rho$ identify with $\leq$, so we have that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}) = e \leq e' = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}')$ implies $\mathsf{E} \preceq_\rho \mathsf{E}'$.

For a more complex type $\rho = \rho_1 \to \cdots \to \rho_n \to o$, $n > 0$, we can easily construct e, as follows:

$$e\, e_1 \cdots e_n = \begin{cases} \mathit{true}, & \text{if there exist } d_1, \ldots, d_n \text{ and ground terms } \mathsf{D}_1, \ldots, \mathsf{D}_n \text{ such that,} \\ & \text{for all } i, d_i \sqsubseteq_{\rho_i} e_i, d_i \rhd_{\rho_i} \mathsf{D}_i \text{ and } \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true} \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

To see that e is monotonic, consider $e_1, \ldots, e_n, e'_1, \ldots, e'_n$, such that $e_1 \sqsubseteq_{\rho_1} e'_1, \ldots, e_n \sqsubseteq_{\rho_n} e'_n$ and observe that $e\, e_1 \cdots e_n = \mathit{true}$ implies $e\, e'_1 \cdots e'_n = \mathit{true}$, due to the transitivity of $\sqsubseteq_{\rho_i}$. We will now show that $e \rhd_\rho \mathsf{E}$, i.e. for all $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ such that $e_1 \rhd_{\rho_1} \mathsf{E}_1, \ldots, e_n \rhd_{\rho_n} \mathsf{E}_n$, it holds $e\, e_1 \cdots e_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{E}_1 \cdots \mathsf{E}_n)$. This is trivial if $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{true}$, since $e_i \sqsubseteq_{\rho_i} e_i$. Let us now examine the case that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$. For the sake of contradiction, assume $e\, e_1 \cdots e_n = \mathit{true}$. Then, by the construction of e, there must exist $d_1, \ldots, d_n$ and $\mathsf{D}_1, \ldots, \mathsf{D}_n$ such that, for all i, $d_i \sqsubseteq_{\rho_i} e_i$, $d_i \rhd_{\rho_i} \mathsf{D}_i$ and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true}$. By the induction hypothesis, we have that $\mathsf{D}_i \preceq_{\rho_i} \mathsf{E}_i$, for all $i \in \{1, \ldots, n\}$. This, by the $\preceq$-Monotonicity Property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ (Theorem 2), yields that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true} \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$, which is obviously a contradiction. Therefore it has to be that $e\, e_1 \cdots e_n = \mathit{false}$.

Finally, in order to prove the second statement and conclude the induction step, we need to show that for all terms $\mathsf{D}_1 \in U_{\mathsf{P},\rho_1}, \ldots, \mathsf{D}_n \in U_{\mathsf{P},\rho_n}$, it holds $\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n \preceq_o \mathsf{E}'\, \mathsf{D}_1 \cdots \mathsf{D}_n$. By the induction hypothesis, there exist $d_1, \ldots, d_n$, such that $d_1 \rhd_{\rho_1} \mathsf{D}_1, \ldots, d_n \rhd_{\rho_n} \mathsf{D}_n$. Because $e \rhd_\rho \mathsf{E}$ and $\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n$ is of type o, we have $e\, d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\, \mathsf{D}_1 \cdots \mathsf{D}_n)$ by definition. Similarly, we also have $e'\, d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}'\, \mathsf{D}_1 \cdots \mathsf{D}_n)$. Moreover, by $e \sqsubseteq_\rho e'$ we have that $e\, d_1 \cdots d_n \sqsubseteq_o e'\, d_1 \cdots d_n$. This yields the desired result, since $\sqsubseteq_o$ identifies with $\preceq_o$. □

The following variation of the Substitution Lemma states that if the building elements of an expression are assigned meanings that are semantic extensions of their syntactic counterparts, then the meaning of the expression is itself a semantic extension of the expression.

Lemma 6. *Let* P *be a program,* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation and I be a Herbrand interpretation of* P*. Also, let* E *be an expression of some argument type* ρ *and let s be a Herbrand state and* θ *be a ground substitution, both with domain vars*(E)*. If, for all predicates* p *of type* π *appearing in* E*,* $[\![\mathsf{p}]\!](I) \rhd_\pi \mathsf{p}$ *and, for all variables* V *of type* ρ' *in vars*(E)*,* $s(\mathsf{V}) \rhd_{\rho'} \theta(\mathsf{V})$*, then* $[\![\mathsf{E}]\!]_s(I) \rhd_\rho \mathsf{E}\theta$.

Proof. The proof is by induction on the structure of E. The basis cases $\mathsf{E} = \mathsf{p}$ and $\mathsf{E} = \mathsf{V}$ hold by assumption and $\mathsf{E} = c : \iota$ is trivial. For the first case of the induction step, let $\mathsf{E} = (\mathsf{f}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$, where $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are of type ι. By the induction hypothesis, we have that $[\![\mathsf{E}_1]\!]_s(I) \rhd_\iota \mathsf{E}_1\theta, \ldots, [\![\mathsf{E}_n]\!]_s(I) \rhd_\iota \mathsf{E}_n\theta$. As $\rhd_\iota$ is defined as equality, we have that $[\![\mathsf{E}]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I)\ \cdots\ [\![\mathsf{E}_n]\!]_s(I) = \mathsf{f}\ \mathsf{E}_1\theta\ \cdots\ \mathsf{E}_n\theta = \mathsf{E}\theta$ and therefore $[\![\mathsf{E}]\!]_s(I) \rhd_\iota \mathsf{E}\theta$. For the second case, let $\mathsf{E} = \mathsf{E}_1\,\mathsf{E}_2$, where E_1 is of type $\rho_1 = \rho_2 \to \pi$ and E_2 is of type ρ_2; then, $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\,[\![\mathsf{E}_2]\!]_s(I)$. By the induction hypothesis, $[\![\mathsf{E}_1]\!]_s(I) \rhd_{\rho_2\to\pi} \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) \rhd_{\rho_2} \mathsf{E}_2\theta$, thus, by definition, $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\ [\![\mathsf{E}_2]\!]_s(I) \rhd_\pi \mathsf{E}_1\theta\ \mathsf{E}_2\theta = (\mathsf{E}_1\,\mathsf{E}_2)\theta = \mathsf{E}\theta$. Finally, we have the case that $\mathsf{E} = (\mathsf{E}_1 \approx \mathsf{E}_2)$, where E_1 and E_2 are both of type ι. The induction hypothesis yields $[\![\mathsf{E}_1]\!]_s(I) \rhd_\iota \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) \rhd_\iota \mathsf{E}_2\theta$ or, since $\rhd_\iota$ is defined as equality, $[\![\mathsf{E}_1]\!]_s(I) = \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) = \mathsf{E}_2\theta$. Then $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$ iff $\mathsf{E}_1\theta = \mathsf{E}_2\theta$ and, equivalently, $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $\mathsf{E}\theta = \mathit{true}$, which implies $[\![\mathsf{E}]\!]_s(I) \rhd_o \mathsf{E}\theta$. □

We are now ready to present the main result of this paper. The theorem establishes the equivalence of Wadge's semantics and Bezem's semantics, in stating that their respective minimum models assign the same meaning to all ground atoms.

Theorem 4. *Let* P *be a program and let* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation. Let* M_P *be the* $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$*-minimum Herbrand model of* P *and let* $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ *be the* $\leq$*-minimum model of* $\mathsf{Gr}(\mathsf{P})$*. Then, for every* $\mathsf{A} \in U_{\mathsf{P},o}$ *it holds* $[\![\mathsf{A}]\!](M_\mathsf{P}) = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{A})$.

Proof. We will construct an interpretation N for P and prove some key properties for this interpretation. Then we will utilize these properties to prove the desired result. The definition of N is as follows:

For every $\mathsf{p} : \rho_1 \to \cdots \to \rho_n \to o$ and all $d_1 \in [\![\rho_1]\!], \ldots, d_n \in [\![\rho_n]\!]$

$$N(\mathsf{p})\,d_1 \cdots d_n = \begin{cases} \mathit{false}, & \text{if there exist } e_1, \ldots, e_n \text{ and ground terms } \mathsf{E}_1, \ldots, \mathsf{E}_n \text{ such that,} \\ & \text{for all } i, d_i \sqsubseteq_{\rho_i} e_i, e_i \rhd_{\rho_i} \mathsf{E}_i \text{ and } \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false} \\ \mathit{true}, & \text{otherwise} \end{cases}$$

Observe that N is a valid Herbrand interpretation of P, in the sense that it assigns elements in $[\![\pi]\!]$ (i.e. functions that are monotonic with respect to $\sqsubseteq_\pi$) to every predicate of type π in P. Indeed, if it was not so, then for some predicate $\mathsf{p} : \pi = \rho_1 \to \cdots \to \rho_n \to o$, there would exist tuples $(d_1, \ldots, d_n)$ and $(d'_1, \ldots, d'_n)$ with $d_1 \sqsubseteq_{\rho_1} d'_1, \ldots, d_n \sqsubseteq_{\rho_n} d'_n$, such that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{true}$ and $N(\mathsf{p})\,d'_1 \cdots d'_n = \mathit{false}$. By definition, the fact that $N(\mathsf{p})\,d'_1 \cdots d'_n$ is assigned the value *false*, would imply that there exist $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ as in the above definition, such that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$ and $d'_1 \sqsubseteq_{\rho_1} e_1, \ldots, d'_n \sqsubseteq_{\rho_n} e_n$. Being that $\sqsubseteq_{\rho_i}$ are transitive relations, the latter yields that $d_1 \sqsubseteq_{\rho_1} e_1, \ldots, d_n \sqsubseteq_{\rho_n} e_n$. Therefore, by definition, $N(\mathsf{p})\,d_1 \cdots d_n$ should also evaluate to *false*, which constitutes a contradiction and thus confirms that the meaning of p is monotonic with respect to $\sqsubseteq_\pi$.

It is also straightforward to see that $N(\mathsf{p}) \rhd_\pi \mathsf{p}$, i.e. for all $d_1, \ldots, d_n$ and all ground terms $\mathsf{D}_1, \ldots, \mathsf{D}_n$ such that $d_1 \rhd_{\rho_1} \mathsf{D}_1, \ldots, d_n \rhd_{\rho_n} \mathsf{D}_n$, we have $N(\mathsf{p})\,d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n)$. Because $d_i \sqsubseteq_{\rho_i} d_i$, this holds trivially if $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{false}$. Now let $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true}$ and assume, for the sake of contradiction, that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{false}$. Then, by the definition of N, there must exist $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ such that, for all i, $d_i \sqsubseteq_{\rho_i} e_i$, $e_i \rhd_{\rho_i} \mathsf{E}_i$ and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$. Thus, by the second part of Lemma 5, for all i, $\mathsf{D}_i \preceq_{\rho_i} \mathsf{E}_i$ and, by the $\preceq$-Monotonicity Property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{D}_1 \cdots \mathsf{D}_n) \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$, which is obviously a contradiction. Thus we conclude that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{true}$.

Next we prove that N is a model of P. Let $\mathsf{p}\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_k$ be a clause in P and let $\{\mathsf{V}_1, \ldots, \mathsf{V}_n, \mathsf{X}_1, \ldots, \mathsf{X}_m\}$, with $\mathsf{V}_i : \rho_i$, for all $i \in \{1, \ldots, n\}$, and $\mathsf{X}_i : \iota$, for all $i \in \{1, \ldots, m\}$, be the set of

variables appearing in the clause. Then, it suffices to show that, for any tuple $(d_1,\ldots,d_n)$ of arguments and any Herbrand state s such that $s(\mathsf{V}_i)=d_i$ for all $i\in\{1,\ldots,n\}$, $N(\mathsf{p})\,d_1\cdots d_n=\mathit{false}$ implies that, for at least one $j\in\{1,\ldots,k\}$, $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$. Again, by the definition of N, we see that if $N(\mathsf{p})\,d_1\cdots d_n=\mathit{false}$, then there exist $e_1,\ldots,e_n$ and ground terms $\mathsf{E}_1,\ldots,\mathsf{E}_n$ such that $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=\mathit{false}$, $d_1\sqsubseteq_{\rho_1}e_1,\ldots,d_n\sqsubseteq_{\rho_n}e_n$ and $e_1\rhd_{\rho_1}\mathsf{E}_1,\ldots,d_n\rhd_{\rho_n}\mathsf{E}_n$. Let θ be a ground substitution such that $\theta(\mathsf{V}_i)=\mathsf{E}_i$ for all $i\in\{1,\ldots,n\}$ and, for all $i\in\{1,\ldots,m\}$, $\theta(\mathsf{X}_i)=s(\mathsf{X}_i)$; then there exists a ground instantiation $\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n\leftarrow\mathsf{B}_1\theta,\ldots,\mathsf{B}_k\theta$ of the above clause in $\mathsf{Gr(P)}$. As $\mathscr{M}_{\mathsf{Gr(P)}}$ is a model of the ground program, $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=\mathit{false}$ implies that there exists at least one $j\in\{1,\ldots,k\}$ such that $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{B}_j\theta)=\mathit{false}$. We are going to show that the latter implies that $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$, which proves that N is a model of P. Indeed, let s' be a Herbrand state such that $s'(\mathsf{V}_i)=e_i\rhd_{\rho_i}\theta(\mathsf{V}_i)=\mathsf{E}_i$ for all $i\in\{1,\ldots,n\}$ and $s'(\mathsf{X}_i)=\theta(\mathsf{X}_i)=s(\mathsf{X}_i)$ for all $i\in\{1,\ldots,m\}$. As we have shown earlier, $N(\mathsf{p}')\rhd_{\pi'}\mathsf{p}'$ for any predicate $\mathsf{p}':\pi'$, thus by Lemma 6 we get $[\![\mathsf{B}_j]\!]_{s'}(N)\rhd_o\mathsf{B}_j\theta$. Since B_j is of type o, the latter reduces to $[\![\mathsf{B}_j]\!]_{s'}(N)=\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{B}_j\theta)=\mathit{false}$. Also, because $d_i\sqsubseteq_{\rho_i}e_i$, i.e. $s\sqsubseteq_{\mathscr{I}_\mathsf{P}}s'$, by the second part of Lemma 3 we get $[\![\mathsf{B}_j]\!]_s(N)\sqsubseteq_o[\![\mathsf{B}_j]\!]_{s'}(N)$, which makes $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$.

Now we can proceed to prove that, for all $\mathsf{A}\in U_{\mathsf{P},o}$, $[\![\mathsf{A}]\!](M_\mathsf{P})=\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{A})$. Let A be of the form $\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n$, where $\mathsf{p}:\rho_1\to\cdots\to\rho_n\to o\in\mathsf{P}$ and let $d_1=[\![\mathsf{E}_1]\!](M_\mathsf{P}),\ldots,d_n=[\![\mathsf{E}_n]\!](M_\mathsf{P})$. As we have shown, N is a Herbrand model of P, while M_P is the minimum, with respect to $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$, of all Herbrand models of P, therefore we have that $M_\mathsf{P}\sqsubseteq_{\mathscr{I}_\mathsf{P}}N$. By definition, this gives us that $M_\mathsf{P}(\mathsf{p})\,d_1\cdots d_n\sqsubseteq_o N(\mathsf{p})\,d_1\cdots d_n$ (1) and, by the first part of Lemma 3, that $d_1\sqsubseteq_{\rho_1}[\![\mathsf{E}_1]\!](N),\ldots,d_n\sqsubseteq_{\rho_n}[\![\mathsf{E}_n]\!](N)$ (2). Moreover, for all predicates $\mathsf{p}':\pi'$ in P, we have $N(\mathsf{p}')\rhd_{\pi'}\mathsf{p}'$ and thus, by Lemma 6, taking s and θ to be empty, we get $[\![\mathsf{E}_i]\!](N)\rhd_{\rho_i}\mathsf{E}_i, 1\le i\le n$. In conjunction with (2), the latter suggests that if $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=\mathit{false}$ then $N(\mathsf{p})\ d_1\ \cdots\ d_n=\mathit{false}$, or, in other words, that $N(\mathsf{p})\ d_1\cdots d_n\le\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$. Because of (1), this makes it that $M_\mathsf{P}(\mathsf{p})\ d_1\cdots d_n\le\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$ (3). On the other hand, by Theorem 3, $M_\mathsf{P}|_{\mathsf{Gr(P)}}$ is a model of $\mathsf{Gr(P)}$ and therefore $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)\le M_\mathsf{P}|_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$, since $\mathscr{M}_{\mathsf{Gr(P)}}$ is the minimum model of $\mathsf{Gr(P)}$. By the definition of $\mathsf{Gr(P)}$ and the meaning of application, the latter becomes $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)\le M_\mathsf{P}|_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=M_\mathsf{P}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=M_\mathsf{P}(\mathsf{p})\ [\![\mathsf{E}_1]\!](M_\mathsf{P})\ \cdots\ [\![\mathsf{E}_n]\!](M_\mathsf{P})=M_\mathsf{P}(\mathsf{p})\ d_1\cdots d_n$. The last relation and (3) can only be true simultaneously, if all the above relations hold as equalities, in particular if $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=[\![\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n]\!](M_\mathsf{P})$. □

5 Discussion

We have considered the two existing extensional approaches to the semantics of higher-order logic programming, and have demonstrated that they coincide for the class of definitional programs. It is therefore natural to wonder whether the two semantic approaches continue to coincide if we extend the class of programs we consider. Unfortunately this is not the case, as we discuss below.

A seemingly mild extension to our source language would be to allow higher-order predicate variables that are not formal parameters of a clause, to appear in its body. Such programs are legitimate under Bezem's semantics (ie., they belong to the hoapata class). Moreover, a recent extension of Wadge's semantics [5] also allows such programs. However, for this extended class of programs the equivalence of the two semantic approaches no longer holds as the following example illustrates.

Example 8. Consider the following extended program:

```
p(a):-Q(a).
```

Following Bezem's semantics, we initially take the ground instantiation of the program, namely:

```
p(a):-p(a).
```

and then compute the least model of the above program which assigns to the atom `p(a)` the value *false*. On the other hand, under the approach in [5], the atom `p(a)` has the value *true* in the minimum Herbrand model of the initial program. This is due to the fact that under the semantics of [5], our initial program reads (intuitively speaking) as follows: "`p(a)` is true if there exists a relation that is true of `a`"; actually, there exists one such relation, namely the set $\{$`a`$\}$. This discrepancy between the two semantics is due to the fact that Wadge's semantics is based on *sets* and not solely on the syntactic entities that appear in the program. □

Future work includes the extension of Bezem's approach to higher-order logic programs with negation. An extension of Wadge's approach for such programs has recently been performed in [4]. More generally, the addition of negation to higher-order logic programming appears to offer an interesting and nontrivial area of research, which we are currently pursuing.

References

[1] Marc Bezem (1999): *Extensionality of Simply Typed Logic Programs*. In Danny De Schreye, editor: *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, MIT Press, pp. 395–410.

[2] Marc Bezem (2001): *An Improved Extensionality Criterion for Higher-Order Logic Programs*. In Laurent Fribourg, editor: *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, *Lecture Notes in Computer Science* 2142, Springer, pp. 203–216, doi:10.1007/3-540-44802-0_15.

[3] Marc Bezem (2002): *Hoapata programs are monotonic*. In: *Proceedings NWPT02, Institute of Cybernetics at TTU, Tallinn*, pp. 18–20.

[4] Angelos Charalambidis, Zoltán Ésik & Panos Rondogiannis (2014): *Minimum Model Semantics for Extensional Higher-order Logic Programming with Negation*. *TPLP* 14(4-5), pp. 725–737, doi:10.1017/S1471068414000313.

[5] Angelos Charalambidis, Konstantinos Handjopoulos, Panagiotis Rondogiannis & William W. Wadge (2013): *Extensional Higher-Order Logic Programming*. *ACM Trans. Comput. Log.* 14(3), p. 21, doi:10.1145/2499937.2499942.

[6] Weidong Chen, Michael Kifer & David Scott Warren (1993): *HILOG: A Foundation for Higher-Order Logic Programming*. *Journal of Logic Programming* 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[7] Vassilis Kountouriotis, Panos Rondogiannis & William W. Wadge (2005): *Extensional Higher-Order Datalog*. In: *Short Paper Proceeding of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pp. 1–5.

[8] John W. Lloyd (1987): *Foundations of Logic Programming*. Springer Verlag, doi:10.1007/978-3-642-83189-8.

[9] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*, 1st edition. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9781139021326.

[10] William W. Wadge (1991): *Higher-Order Horn Logic Programming*. In Vijay A. Saraswat & Kazunori Ueda, editors: *Logic Programming, Proceedings of the 1991 International Symposium, San Diego, California, USA, Oct. 28 - Nov 1, 1991*, MIT Press, pp. 289–303.

Formalizing Termination Proofs under Polynomial Quasi-interpretations

Naohi Eguchi*
Department of Mathematics and Informatics
Chiba University, Japan
neguchi@g.math.s.chiba-u.ac.jp

Usual termination proofs for a functional program require to check all the possible reduction paths. Due to an exponential gap between the height and size of such the reduction tree, no naive formalization of termination proofs yields a connection to the polynomial complexity of the given program. We solve this problem employing the notion of minimal function graph, a set of pairs of a term and its normal form, which is defined as the least fixed point of a monotone operator. We show that termination proofs for programs reducing under lexicographic path orders (LPOs for short) and polynomially quasi-interpretable can be optimally performed in a weak fragment of Peano arithmetic. This yields an alternative proof of the fact that every function computed by an LPO-terminating, polynomially quasi-interpretable program is computable in polynomial space. The formalization is indeed optimal since every polynomial-space computable function can be computed by such a program. The crucial observation is that inductive definitions of minimal function graphs under LPO-terminating programs can be approximated with transfinite induction along LPOs.

1 Introduction

1.1 Motivation

The termination of a program states that any reduction under the program leads to a normal form. Recent developments in termination analysis of first order functional programs, or of *term rewrite systems* more specifically, have drawn interest in computational resource analysis, i.e., not just the termination but also the estimation of time/space-resources required to execute a given program, which includes the polynomial run-space complexity analysis. Usual termination proofs for a program require to check all the possible reduction paths under the program. Due to an exponential gap between the *height* and *size* of such the reduction tree, no naive termination proof yields a connection to the polynomial complexity of the given program. For the sake of optimal termination proofs, it seems necessary to discuss "all the possible reduction paths" by means of an alternative notion smaller in size than reduction trees.

1.2 Backgrounds

Stemming from [21], there are various functional characterizations of polynomial-space computable functions [14, 16, 17, 9], Those characterizations state that every poly-space computable function can be defined by a finite set of equations, i.e., by a functional program. Orienting those equations suitably, such programs reduce under a termination order, the *lexicographic path orders* (LPOs for short). The well-founded-ness of LPOs yields the termination of the reducing programs.

*The author is supported by Grants-in-Aid for JSPS Fellows (Grant No. 25 · 726).

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 33–47, doi:10.4204/EPTCS.191.5

In the seminal work [5], it was discussed, depending on the choice of a termination order, what mathematical axiom is necessary to formalize termination proofs by the termination order within Peano arithmetic PA that axiomatizes ordered semi-rings with mathematical induction. In case of *multiset path orders* (MPOs for short), termination proofs can be formalized in the fragment of PA with induction restricted to computably enumerable sets. This yields an alternative proof of the fact that every function computed by an MPO-terminating program is primitive recursive, cf. [10]. The formalization is optimal since every primitive recursive function can be computed by an MPO-terminating program. In case of LPOs, termination proofs can be formalized in the fragment with induction restricted to expressions of the form "f is total" for some computable function f. The formalization is optimal in the same sense as in case of MPOs, cf. [22].

In more recent works [3, 4], MPOs and LPOs are combined with *polynomial quasi-interpretations* (PQIs for short). Unlike (strict) polynomial interpretations [2], the existence of a quasi-interpretation does not tell us anything about termination. However, combined with these termination orders, the PQI can be a powerful method in computational resource analysis. Indeed, those functional programs characterizing poly-space computable functions that was motioned above admit PQIs. This means that every poly-space computable function can be computed by an LPO-terminating program that admits a PQI. Moreover, conversely, every function computed by such a program is computable in polynomial space [3, Theorem 1].

1.3 Outline

In Section 2 we fix the syntax of first order functional programs and the semantics in accordance with the syntax. In Section 3 we present the definitions of LPOs and PQIs together with some examples, stating an application to poly-space computable functions (Theorem 1, [3, Theorem 1]). In Section 4 we present the framework of formalization. For an underlying formal system, a second order system U^1_2 of *bounded arithmetic* [6], which can be regarded as a weak fragment of PA, seems suitable since it is known that the system U^1_2 is complete for poly-space computable functions (Theorem 2.2).

In [5], the termination of a program reducing under an LPO $<_{\mathsf{lpo}}$ is deduced by showing that, given a term t, a tree containing all the possible reduction chains starting with t is well founded under $<_{\mathsf{lpo}}$. The same construction of such reduction trees does not work in U^1_2 essentially because the exponentiation $m \mapsto 2^m$ is not available. We lift the problem employing the notion of *minimal function graph* [12, 11, 15], a set of pairs of a term and its normal form. Given a term t, instead of constructing a reduction tree rooted at t, we construct a (subset of a) minimal function graph that stores the pair of t and a normal form of t. Typically, a minimal function graph is inductively defined, or in other words defined as the least fixed point of a monotone operator. Let us recall that the set of natural numbers is the least fixed point of the operator $m \in \Gamma(X) \Longleftrightarrow m = 0 \vee \exists n \in X$ s.t. $m = n+1$. As seen from this example, many instances of inductive definitions are induced by operators of the form $t \in \Gamma(X) \Longleftrightarrow \exists s_1, \ldots, s_k \in X \cdots$. Crucially, a minimal function graph under a program reducing under an LPO $<_{\mathsf{lpo}}$ can be defined as the least fixed point of such an operator but also $t \in \Gamma(X) \Longleftrightarrow \exists s_1, \ldots, s_k \in X \wedge s_1, \ldots, s_k <_{\mathsf{lpo}} t \cdots$ holds. Thanks to the additional condition $s_1, \ldots, s_k <_{\mathsf{lpo}} t$, the minimal function graphs under the program can be defined by $<_{\mathsf{lpo}}$-transfinite induction as well as inductive definitions. In Section 5 this idea is discussed in more details.

In the main section, Section 6, the full details about the formalization are given. Most of the effort is devoted to deduce in U^1_2 an appropriate form of transfinite induction along LPOs (Lemma 5). Based on the idea above, we then construct a minimal function graph G for a given program $\mathbf{R}$ reducing under an LPO $<_{\mathsf{lpo}}$ by $<_{\mathsf{lpo}}$-transfinite induction (Theorem 3). Since G stores all the pairs of a term and its

R-normal form, this means the termination of the program **R**.

In Section 7 it is shown that the formalization presented in Section 6 yields that every function computed by an LPO-terminating program that admits a PQI is poly-space computable (Corollary 3). This shows that the formalization is optimal since such programs can only compute poly-space computable functions as mentioned in Section 1.2.

2 Syntax and semantics of first order functional programs

Throughout the paper, a *program* denotes a *term rewrite system*. We sometimes use unusual notations or formulations for the sake of simplification. More precise, widely accepted formulations can be found, e.g., in [20].

Definition 1 (Constuctor-, basic-, terms, rewrite rules, sizes of terms)**.** Let $\mathbf{C}$ and $\mathbf{D}$ be disjoint finite signatures, respectively of *constructors* and *defined* symbols, and $\mathbf{V}$ a countably infinite set of *variables*. We assume that $\mathbf{C}$ contains at least one constant. The sets $\mathbf{T}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$ of *terms*, $\mathbf{T}(\mathbf{C},\mathbf{V})$ of *constructor* terms, $\mathbf{B}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$ of *basic* terms and $\mathbf{R}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$ of *rewrite rules* are distinguished as follows.

$$\begin{array}{lrcll}
\text{(Terms)} & t & ::= & x \mid c(t_1,\ldots,t_l) \mid f(t_1,\ldots,t_l) & \in \mathbf{T}(\mathbf{C}\cup\mathbf{D},\mathbf{V});\\
\text{(Constructor terms)} & s & ::= & x \mid c(s_1,\ldots,s_k) & \in \mathbf{T}(\mathbf{C},\mathbf{V});\\
\text{(Basic terms)} & u & ::= & f(s_1,\ldots,s_k) & \in \mathbf{B}(\mathbf{C}\cup\mathbf{D},\mathbf{V});\\
\text{(Rewrite rules)} & \rho & ::= & u\to t & \in \mathbf{R}(\mathbf{C}\cup\mathbf{D},\mathbf{V}),
\end{array}$$

where $x\in\mathbf{V}$, $c\in\mathbf{C}$, $f\in\mathbf{D}$, $t,t_1,\ldots,t_l\in\mathbf{T}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$, $s_1,\ldots,s_k\in\mathbf{T}(\mathbf{C},\mathbf{V})$ and $u\in\mathbf{B}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$. For such a class $\mathbf{S}(\mathbf{F},\mathbf{V})$ of terms, $\mathbf{S}(\mathbf{F})$ denotes the subset of closed terms. The *size* $\|t\|$ of a term t is defined as $\|x\|=1$ for a variable x and $\|f(t_1,\ldots,t_k)\|=1+\sum_{j=1}^{k}\|t_j\|$.

Definition 2 (Substitutions, quasi-reducible programs, rewrite relations)**.** A *program* $\mathbf{R}$ is a finite subset of $\mathbf{R}(\mathbf{C}\cup\mathbf{D},\mathbf{V})$ consisting of rewrite rules of the form $l\to r$ such that the variables occurring in r occur in l as well. A mapping $\theta:\mathbf{V}\to\mathbf{S}(\mathbf{F},\mathbf{V})$ from variables to a set $\mathbf{S}(\mathbf{F},\mathbf{V})$ of terms is called a *substitution*. For a term $t\in\mathbf{S}(\mathbf{F},\mathbf{V})$, $t\theta$ denotes the result of replacing every variable x with $\theta(x)$. A program $\mathbf{R}$ is *quasi-reducible* if, for any closed basic term $t\in\mathbf{B}(\mathbf{C}\cup\mathbf{F})$, there exist a rule $l\to r\in\mathbf{R}$ and a substitution $\theta:\mathbf{V}\to\mathbf{T}(\mathbf{C})$ such that $t=l\theta$. We restrict reductions to those under *call-by-value* evaluation, or *innermost* reductions more precisely. For three terms t,u,v, we write $t[u/v]$ to denote the result of replacing an occurrence of v with u. It will not be indicated which occurrence of v is replaced if no confusion likely arises. We write $t\xrightarrow[\mathbf{R}]{\mathrm{i}}s$ if $s=t[r\theta/l\theta]$ holds for some rule $l\to r\in\mathbf{R}$ and constructor substitution $\theta:\mathbf{V}\to\mathbf{T}(\mathbf{C})$. We write $\xrightarrow[\mathbf{R}]{\mathrm{i}}{}^{*}$ to denote the reflexive and transitive closure of $\xrightarrow[\mathbf{R}]{\mathrm{i}}$ and $t\xrightarrow[\mathbf{R}]{\mathrm{i}}{}^{!}s$ if $t\xrightarrow[\mathbf{R}]{\mathrm{i}}{}^{*}s$ and s is a normal form. By definition, for any quasi-reducible program $\mathbf{R}$, if $t\xrightarrow[\mathbf{R}]{\mathrm{i}}{}^{!}s$ and t is closed, then $s\in\mathbf{T}(\mathbf{C})$ holds.

A program $\mathbf{R}$ *computes* a function if any closed basic term has a unique normal form in $\mathbf{T}(\mathbf{C})$. In this case, for every k-ary function symbol $f\in\mathbf{D}$, a function $[\![f]\!]:\mathbf{T}(\mathbf{C})^k\to\mathbf{T}(\mathbf{C})$ is defined by $[\![f]\!](s_1,\ldots,s_k)=s\iff f(s_1,\ldots,s_k)\xrightarrow[\mathbf{R}]{\mathrm{i}}{}^{!}s$.

3 Lexicographic path orders and quasi-interpretations

Lexicographic path orders are *recursive path orders* with lexicographic status only, whose variant was introduced in [13]. Recursive path orders with multiset status only were introduced in [8] and a modern formulation with both multiset and lexicographic status can be found in [20, page 211]. Let $<_{\mathbf{F}}$ be a

(strict) *precedence*, a well-founded partial order on a signature $\mathbf{F} = \mathbf{C} \cup \mathbf{D}$. We always assume that every constructor is $<_{\mathbf{F}}$-minimal. The *lexicographic path order* (LPO for short) $<_{\mathsf{lpo}}$ induced by $<_{\mathbf{F}}$ is defined recursively by the following three rules.

1. $\dfrac{s \leqslant_{\mathsf{lpo}} t_i}{s <_{\mathsf{lpo}} g(t_1,\ldots,t_l)}$ $(i \in \{1,\ldots,l\})$
2. $\dfrac{s_1 <_{\mathsf{lpo}} g(t_1,\ldots,t_l) \quad \cdots \quad s_k <_{\mathsf{lpo}} g(t_1,\ldots,t_l)}{f(s_1,\ldots,s_k) <_{\mathsf{lpo}} g(t_1,\ldots,t_l)}$ $(f <_{\mathbf{F}} g \in \mathbf{D})$
3. $\dfrac{s_1 = t_1 \ \cdots \ s_{i-1} = t_{i-1} \quad s_i <_{\mathsf{lpo}} t_i \quad s_{i+1} <_{\mathsf{lpo}} t \ \cdots \ s_k <_{\mathsf{lpo}} t}{f(s_1,\ldots,s_k) <_{\mathsf{lpo}} f(t_1,\ldots,t_k) = t}$ $(f \in \mathbf{D})$

We say that a program $\mathbf{R}$ *reduces under* $<_{\mathsf{lpo}}$ if $r <_{\mathsf{lpo}} l$ holds for each rule $l \to r \in \mathbf{R}$ and that $\mathbf{R}$ is *LPO-terminating* if there exists an LPO under which $\mathbf{R}$ reduces. We write $s <^{\langle i \rangle}_{\mathsf{lpo}} t$ if $s <_{\mathsf{lpo}} t$ results as an instance of the above i^{th} case ($i = 1,2,3$). Corollary 1 is a consequence of the definition of LPOs, following from $<_{\mathbf{F}}$-minimality of constructors.

Corollary 1. *If $s <_{\mathsf{lpo}} t$ and $t \in \mathbf{T}(\mathbf{C})$, then $s <^{\langle 1 \rangle}_{\mathsf{lpo}} t$ and $s \in \mathbf{T}(\mathbf{C})$.*

A *quasi-interpretation* $(\!|\cdot|\!)$ for a signature $\mathbf{F}$ is a mapping from $\mathbf{F}$ to functions over naturals fulfilling (i) $(\!|f|\!) : \mathbb{N}^k \to \mathbb{N}$ for each k-ary function symbol $f \in \mathbf{F}$, (ii) $(\!|f|\!)(\ldots,m,\ldots) \leq (\!|f|\!)(\ldots,n,\ldots)$ whenever $m < n$, (iii) $m_j \leq (\!|f|\!)(m_1,\ldots,m_k)$ for any $j \in \{1,\ldots,k\}$, and (iv) $0 < (\!|f|\!)$ if f is a constant. A quasi-interpretation $(\!|\cdot|\!)$ for a signature $\mathbf{F}$ is extended to closed terms $\mathbf{T}(\mathbf{F})$ by $(\!|f(t_1,\ldots,t_k)|\!) = (\!|f|\!)((\!|t_1|\!),\ldots,(\!|t_k|\!))$. Such an interpretation $(\!|\cdot|\!)$ is called a quasi-interpretation for a program $\mathbf{R}$ if $(\!|r\theta|\!) \leq (\!|l\theta|\!)$ holds for each rule $l \to r \in \mathbf{R}$ and for any constructor substitution $\theta : \mathbf{V} \to \mathbf{T}(\mathbf{C})$. A program $\mathbf{R}$ *admits a polynomial quasi-interpretation* (PQI for short) if there exists a quasi-interpretation $(\!|\cdot|\!)$ for $\mathbf{R}$ such that $(\!|f|\!)$ is polynomially bounded for each $f \in \mathbf{F}$. A PQI $(\!|\cdot|\!)$ is called *kind* 0 (or *additive* [4]) if, for each constructor $c \in \mathbf{C}$, $(\!|c|\!)(m_1,\ldots,m_k) = d + \sum_{j=1}^{k} m_j$ holds for some constant $d > 0$. An LPO-terminating program $\mathbf{R}$ is called an $LPO^{Poly(0)}$-program if $\mathbf{R}$ admits a kind 0 PQI.

Theorem 1 ([3])**.** *Every function computed by an $LPO^{Poly(0)}$-program is computable in polynomial space.*

Conversely, every polynomial-space computable function can be computed by an $\mathrm{LPO}^{\mathrm{Poly}(0)}$-program [3, Theorem 1]. In [4] various examples of programs admitting (kind 0) PQIs are illustrated, including $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs $\mathbf{R}_{\mathsf{lcs}}$ and $\mathbf{R}_{\mathsf{QBF}}$ below.

Example 1. The length of the *longest common subsequences* of two strings can be computed by a program $\mathbf{R}_{\mathsf{lcs}}$ [4, Example 6], which consists of the following rewrite rules defined over a signature $\mathbf{F} = \mathbf{C} \cup \mathbf{D}$ where $\mathbf{C} = \{0,\mathsf{s},\varepsilon,\mathsf{a},\mathsf{b}\}$ and $\mathbf{D} = \{\mathsf{max},\mathsf{lcs}\}$.

$$
\begin{array}{llll}
\mathsf{max}(x,0) \to x & \mathsf{max}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{max}(x,y)) & & \\
\mathsf{max}(0,y) \to y & & & \\
\mathsf{lcs}(x,\varepsilon) \to 0 & \mathsf{lcs}(\mathsf{i}(x),\mathsf{i}(y)) \to \mathsf{s}(\mathsf{lcs}(x,y)) & & (\mathsf{i} \in \{\mathsf{a},\mathsf{b}\}) \\
\mathsf{lcs}(\varepsilon,y) \to 0 & \mathsf{lcs}(\mathsf{i}(x),\mathsf{j}(y)) \to \mathsf{max}(\mathsf{lcs}(x,\mathsf{j}(y)),\mathsf{lcs}(\mathsf{i}(x),y)) & & (\mathsf{i} \neq \mathsf{j} \in \{\mathsf{a},\mathsf{b}\})
\end{array}
$$

Natural numbers are built of 0 and s and strings of a and b as $\mathsf{a}(u) = \mathsf{a}u$ for a string $u \in \{\mathsf{a},\mathsf{b}\}^*$. The symbol ε denotes the empty string. Define a precedence $<_{\mathbf{F}}$ on $\mathbf{F}$ by $\mathsf{max} <_{\mathbf{F}} \mathsf{lcs}$. Assuming that every constructor is $<_{\mathbf{F}}$-minimal, the program $\mathbf{R}_{\mathsf{lcs}}$ reduces under the LPO $<_{\mathsf{lpo}}$ induced by $<_{\mathbf{F}}$. For instance, the orientation $\mathsf{max}(\mathsf{lcs}(x,\mathsf{b}(y)),\mathsf{lcs}(\mathsf{a}(x),y)) <_{\mathsf{lpo}} \mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))$ can be deduced as follows. The orientation $y <^{\langle 1 \rangle}_{\mathsf{lpo}} \mathsf{b}(y)$ yields $\mathsf{lcs}(\mathsf{a}(x),y) <^{\langle 3 \rangle}_{\mathsf{lpo}} \mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))$ while $x <^{\langle 1 \rangle}_{\mathsf{lpo}} \mathsf{a}(x)$ and $\mathsf{b}(y) <^{\langle 1 \rangle}_{\mathsf{lpo}} \mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))$ yield

$\mathsf{lcs}(x,\mathsf{b}(y)) <^{\langle 3\rangle}_{\mathsf{lpo}} \mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))$. These together with $\mathsf{max} <_{\mathbf{F}} \mathsf{lcs}$ yield $\mathsf{max}(\mathsf{lcs}(x,\mathsf{b}(y)),\mathsf{lcs}(\mathsf{a}(x),y)) <^{\langle 2\rangle}_{\mathsf{lpo}} \mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))$. It can be seen that the program $\mathbf{R}_{\mathsf{lcs}}$ admits the kind 0 PQI $(\!|\cdot|\!)$ defined by

$$\begin{aligned}(\!|\mathsf{0}|\!) = (\!|\varepsilon|\!) &= 1,\\ (\!|\mathsf{s}|\!)(x) = (\!|\mathsf{a}|\!)(x) = (\!|\mathsf{b}|\!)(x) &= 1+x,\\ (\!|\mathsf{max}|\!)(x,y) = (\!|\mathsf{lcs}|\!)(x,y) &= \max(x,y).\end{aligned}$$

This is exemplified as $(\!|\mathsf{max}(\mathsf{lcs}(x,\mathsf{b}(y)),\mathsf{lcs}(\mathsf{a}(x),y))|\!) = \max\big(\max(x,1+y),\max(1+x,y)\big) \le \max(1+x,1+y) = (\!|\mathsf{lcs}(\mathsf{a}(x),\mathsf{b}(y))|\!)$. Thus Theorem 1 implies that the function $[\![\mathsf{lcs}]\!]$ can be computed in polynomial space.

Example 2. The *Quantified Boolean Formula* (QBF) problem can be solved by a program $\mathbf{R}_{\mathsf{QBF}}$ [4, Example 36], which consists of the following rewrite rules defined over a signature $\mathbf{F} = \mathbf{C} \cup \mathbf{D}$ where $\mathbf{C} = \{\mathsf{0},\mathsf{s},\mathsf{nil},\mathsf{cons},\top,\bot,\mathsf{var},\neg,\vee,\exists\}$ and $\mathbf{D} = \{=,\mathsf{not},\mathsf{or},\mathsf{in},\mathsf{verify},\mathsf{qbf}\}$.

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{not}(\top) & \to & \bot & \mathsf{not}(\bot) & \to & \top\\
\mathsf{or}(\top,x) & \to & \top & \mathsf{or}(\bot,x) & \to & x\\
\mathsf{0}=\mathsf{0} & \to & \top & \mathsf{s}(x)=\mathsf{0} & \to & \bot\\
\mathsf{0}=\mathsf{s}(x) & \to & \bot & \mathsf{s}(x)=\mathsf{s}(y) & \to & x=y\\
\mathsf{in}(x,\mathsf{nil}) & \to & \bot & \mathsf{in}(x,\mathsf{cons}(y,ys)) & \to & \mathsf{or}(x=y,\mathsf{in}(x,ys))
\end{array}$$

$$\begin{array}{rcl}
\mathsf{verify}(\mathsf{var}(x),xs) & \to & \mathsf{in}(x,xs)\\
\mathsf{verify}(\neg x,xs) & \to & \mathsf{not}(\mathsf{verify}(x,xs))\\
\mathsf{verify}(x\vee y,xs) & \to & \mathsf{or}\,(\mathsf{verify}(x,xs),\mathsf{verify}(y,xs))\\
\mathsf{verify}\,((\exists x)y,xs) & \to & \mathsf{or}\,(\mathsf{verify}(y,\mathsf{cons}(x,xs)),\mathsf{verify}\,(y,xs))\\
\mathsf{qbf}(x) & \to & \mathsf{verify}(x,\mathsf{nil})
\end{array}$$

The symbol $\top$ denotes the true Boolean value while $\bot$ the false one. Boolean variables are encoded with $\{\mathsf{0},\mathsf{s}\}$-terms, i.e., with naturals. Formulas are built from variables operating var, $\neg$, $\vee$ or $\exists$. Without loss of generality, we can assume that every QBF is built up in this way. As usual, terms of the forms $=(s,t)$, $\neg(t)$, $\vee(s,t)$ and $\exists(s,t)$ are respectively denoted as $s=t$, $\neg t$, $s\vee t$ and $(\exists s)t$. By definition, for a Boolean formula φ with Boolean variables $x_1,\ldots,x_k$, $[\![\mathsf{verify}]\!](\varphi,[\cdots]) = \top$ holds if and only if φ is true with the truth assignment that $x_j = \top$ if x_j appears in the list $[\cdots]$ and $x_j = \bot$ otherwise.

Define a precedence $<_{\mathbf{F}}$ over $\mathbf{F}$ by $\mathsf{not},\mathsf{or},= <_{\mathbf{F}} \mathsf{in} <_{\mathbf{F}} \mathsf{verify} <_{\mathbf{F}} \mathsf{qbf}$. Assuming $<_{\mathbf{F}}$-minimality of constructor, the program $\mathbf{R}_{\mathsf{QBF}}$ reduces under the LPO $<_{\mathsf{lpo}}$ induced by $<_{\mathbf{F}}$. For instance, the orientation $\mathsf{or}(\mathsf{verify}(y,\mathsf{cons}(x,xs)),\mathsf{verify}(y,xs)) <_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$ can be deduced as follows. As well as $xs <^{\langle 1\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$, the orientation $x <^{\langle 1\rangle}_{\mathsf{lpo}} \exists(x,y)$ yields $x <^{\langle 1\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$. These together with the assumption $\mathsf{cons} <_{\mathbf{F}} \mathsf{verify}$ yield $\mathsf{cons}(x,xs) <^{\langle 2\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$. This together with $y <^{\langle 1\rangle}_{\mathsf{lpo}} \exists(x,y)$ yields $\mathsf{verify}(y,\mathsf{cons}(x,xs)) <^{\langle 3\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$ as well as $\mathsf{verify}(y,xs) <^{\langle 3\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$. These orientations together with the assumption $\mathsf{or} <_{\mathbf{F}} \mathsf{verify}$ now allow us to deduce the desired orientation $\mathsf{or}(\mathsf{verify}(y,\mathsf{cons}(x,xs)),\mathsf{verify}(y,xs)) <^{\langle 2\rangle}_{\mathsf{lpo}} \mathsf{verify}(\exists(x,y),xs)$.

Furthermore, let us define a PQI $(\!|\cdot|\!)$ for the signature $\mathbf{F}$ by

$$\begin{aligned}
(\!|c|\!) &= 1 && \text{if } c \text{ is a constant,}\\
(\!|x_1,\ldots,x_k|\!) &= 1+\textstyle\sum_{j=1}^{k} x_j && \text{if } c \in \mathbf{C} \text{ with arity} > 0,\\
(\!|f|\!)(x_1,\ldots,x_k) &= \max_{j=1}^{k} x_j && \text{if } f \in \mathbf{D} \setminus \{\mathsf{verify},\mathsf{qbf}\},\\
(\!|\mathsf{verify}|\!)(x,y) &= x+y,\\
(\!|\mathsf{qbf}|\!)(x) &= x+1.
\end{aligned}$$

Clearly the PQI $(\!|\cdot|\!)$ is kind 0. Then the program $\mathbf{R}_{\mathsf{QBF}}$ admits the PQI. This is exemplified by the rule above as $(\!|\mathsf{or}(\mathsf{verify}(y,\mathsf{cons}(x,xs)),\mathsf{verify}(y,xs))|\!) = \max\big(y+(1+x+xs),y+xs\big) = (1+x+y)+xs = (\!|\mathsf{verify}(\exists(x,y),xs)|\!)$. Thus Theorem 1 implies that the function $[\![\mathsf{qbf}]\!]$ can be computed in polynomial space. This is consistent with the well known fact that the QBF problem is PSPACE-complete.

4 A system U_2^1 of second order bounded arithmetic

In this section, we present the basics of second order bounded arithmetic following [1]. The original formulation is traced back to [6]. The non-logical language $\mathbf{L}_{\mathsf{BA}}$ of first order bounded arithmetic consists of the constant 0, the successor S, the addition $+$, the multiplication $\cdot$, $|x| = \lceil \log_2(x+1) \rceil$, the division by two $\lfloor x/2 \rfloor$, the smash $\#(x,y) = 2^{|x|\cdot|y|}$ and $\leq$. It is easy to see that $|m|$ is equal to the number of bits in the binary representation of a natural m. In addition to these usual symbols, we assume that the language $\mathbf{L}_{\mathsf{BA}}$ contains $\max(x,y)$. The assumption makes no change if an underlying system is sufficiently strong.

Definition 3 (Sharply-, bounded quantifiers, bounded formulas, S_2^1)**.** Quantifiers of the form $\exists x(x \leq t \wedge \cdots)$ or $\forall x(x \leq t \to \cdots)$ for some term t are called *bounded* and quantifiers of the form $(Qx \leq |t|)\cdots$ are called *sharply* bounded. *Bounded formulas* contain no unbounded first order quantifiers. The classes Σ_i^{b} ($i \in \mathbb{N}$) of bounded formulas are defined by counting the number of alternations of bounded quantifiers starting with an existential one, but ignoring sharply bounded ones. For each $i \in \mathbb{N}$, the first order system S_2^i of bounded arithmetic is axiomatized with a set BASIC of open axioms defining the $\mathbf{L}_{\mathsf{BA}}$-symbols together with the schema (Σ_i^{b}-PIND) of bit-wise induction for Σ_i^{b}-formulas.

$$\varphi(0) \wedge \forall x\big(\varphi(\lfloor x/2 \rfloor) \to \varphi(x)\big) \to \forall x \varphi(x) \quad (\varphi \in \Phi) \tag{Φ-PIND}$$

The precise definition of the basic axioms BASIC can be found, e.g., in [7, page 101].

Definition 4 (Second order bounded formulas, U_2^1)**.** In addition to the first order language, the language of second order bounded arithmetic contains second order variables $X, Y, Z, \ldots$ ranging over sets and the membership relation $\in$. In contrast to the classes Σ_i^{b}, the classes $\Sigma_i^{\mathrm{b},1}$ of second order bounded formulas are defined by counting alternations of second order quantifiers starting with an existential one, but ignoring first order ones. By definition, $\Sigma_0^{\mathrm{b},1}$ is the class of bounded formulas with no second order quantifiers. The second order system U_2^1 is axiomatized with BASIC, ($\Sigma_1^{\mathrm{b},1}$-PIND) and the axiom ($\Sigma_0^{\mathrm{b},1}$-CA) of comprehension for $\Sigma_0^{\mathrm{b},1}$-formulas.

$$\forall \vec{x}\, \forall \vec{X}\, \exists Y (\forall y \leq t)\big(y \in Y \leftrightarrow \varphi(y, \vec{x}, \vec{X})\big) \quad (\varphi \in \Phi) \tag{Φ-CA}$$

Unlike first order ones, second order quantifiers have no explicit bounding. However, due to the presence of a bounding term t in the schema ($\Sigma_0^{\mathrm{b},1}$-CA), one can only deduce the existence of a set with a bounded domain.

Example 3. The axiom ($\Sigma_0^{\mathrm{b},1}$-CA) of comprehension allows us to transform given sets $\vec{X}$ into another set Y via $\Sigma_0^{\mathrm{b},1}$-definable operations without inessential encodings. For an easy example, assume that two sets U and V encode binary strings respectively of length m and n in such a way that $j \in U \Leftrightarrow$ "the j^{th} bit of the string U is 1" and $j \notin U \Leftrightarrow$ "the j^{th} bit of the string U is 0" for each $j < m$. Then the *concatenation* $W = U^\frown V$, the string U followed by V, is defined by ($\Sigma_0^{\mathrm{b},1}$-CA) as follows.

$$(\forall j < m+n)\big[j \in W \leftrightarrow \big((j < m \wedge j \in U) \vee (m \leq j \wedge j - m \in V)\big)\big]$$

Definition 5 (Definable functions in formal systems)**.** Let T be one of the formal systems defined above and Φ be a class of bounded formulas. A function $f : \mathbb{N}^k \to \mathbb{N}$ is Φ-*definable in* T if there exists a formula $\varphi(x_1,\ldots,x_k,y) \in \Phi$ with no other free variables such that $\varphi(\vec{x},y)$ expresses the relation $f(\vec{x}) = y$ (under the standard semantics) and T proves the sentence $\forall \vec{x}\, \exists! y \varphi(\vec{x},y)$.

Theorem 2 ([6])**.** *1. A function is* Σ^{b}_1*-definable in* S^1_2 *if and only if it is computable in polynomial time.*

2. A function is $\Sigma^{\mathrm{b},1}_1$*-definable in* U^1_2 *if and only if it is computable in polynomial space.*

To readers who are not familiar with second order bounded arithmetic, it might be of interest to outline the proof that every polynomial-space computable function can be defined in U^1_2. The argument is commonly known as the *divide-and-conquer* method, which was originally used to show the classical inclusion NPSPACE $\subseteq$ PSPACE [18].

Proof of the "if" direction of Theorem 2.2 (Outline). Suppose that a function $f : \mathbb{N}^k \to \mathbb{N}$ is computable in polynomial space. This means that there exist a deterministic Turing machine M and a polynomial $p : \mathbb{N}^k \to \mathbb{N}$ such that, for any inputs $m_1,\ldots,m_k$, $f(m_1,\ldots,m_k)$ can be computed by M while the head of M only visits a number of cells bounded by $p(|m_1|,\ldots,|m_k|)$. Then, since the number of possible configurations under M on inputs $m_1,\ldots,m_k$ is bounded by $2^{q(|m_1|,\ldots,|m_k|)}$ for some polynomial q, the computation terminates in a step bounded by $2^{q(|\vec{m}|)}$ as well.

Let $\psi_M(m_1,\ldots,m_k,n,w_0,W)$ denote a $\Sigma^{\mathrm{b},1}_0$-formula expressing that the set W encodes the concatenation $w_1{}^\frown \cdots {}^\frown w_{2^{|n|}}$ of configurations under M, where w_j is the next configuration of w_{j-1}, writing $w_j = \mathsf{Next}_M(w_{j-1})$ $(1 \le j \le 2^{|n|})$. Reasoning informally in U^1_2, the $\Sigma^{\mathrm{b},1}_1$-formula $\varphi(\vec{m},n) :\equiv (\forall w \le 2^{p(|\vec{m}|)}) \exists W \psi_M(\vec{m},n,w,W)$ can be deduced by ($\Sigma^{\mathrm{b},1}_1$-PIND) on n. In case $n = 0$, W can be defined identical to $\mathsf{Next}_M(w)$. For the induction step, given a configuration $w_0 \le 2^{p(|\vec{m}|)}$, the induction hypothesis yields a set U such that $\psi_M(\vec{m},\lfloor n/2 \rfloor, w_0, U)$ holds. Another instance of the induction hypothesis yields a set V such that $\psi_M(\vec{m},\lfloor n/2 \rfloor, w_{2^{|n|-1}}, V)$ holds. Since $2^{|n|} = 2^{|n|-1} + 2^{|n|-1}$, $\psi_M(\vec{m},n,w_0,W)$ holds for the set $W := U{}^\frown V$.

Now instantiating n with $2^{q(|\vec{m}|)}$ yields a set W such that $\psi_M(\vec{m},2^{q(|\vec{m}|)},\mathsf{Init}_M(\vec{m}),W)$ holds for the initial configuration $\mathsf{Init}_M(\vec{m})$ on inputs $\vec{m}$. The set W yields the final configuration and thus the result $f(\vec{m})$ of the computation. The uniqueness of the result can be deduced in U^1_2 accordingly. □

The "only if" direction of Theorem 2.2 follows from a bit more general statement.

Lemma 1. *If* U^1_2 *proves* $\exists y \varphi(x_1,\ldots,x_k,y)$ *for a* $\Sigma^{\mathrm{b},1}_1$*-formula* $\varphi(x_1,\ldots,x_k,y)$ *with no other free variables, then there exists a function* $f : \mathbb{N}^k \to \mathbb{N}$ *such that, for any naturals* $\vec{m} = m_1,\ldots,m_k \in \mathbb{N}$, (i) $f(\vec{m})$ *is computable with the use of space bounded by a polynomial in* $|m_1|,\ldots,|m_k|$*, and* (ii) $\varphi(\underline{\vec{m}}, \underline{f(\vec{m})})$ *holds under the standard semantics, where* $\underline{m}$ *denotes the numeral* $\mathrm{S}^m(0)$ *for a natural* m*.*

It is also known that the second order system axiomatized with the schema ($\Sigma^{\mathrm{b},1}_1$-IND), instead of ($\Sigma^{\mathrm{b},1}_1$-PIND), of the usual induction $\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(\mathrm{S}(x))) \to \forall x \varphi(x)$ for $\Sigma^{\mathrm{b},1}_1$-formulas, called V^1_2, captures the exponential-time computable functions of polynomial growth rate in the sense of Theorem 2. Though there is no common notion about what is bounded arithmetic, the exponential function $m \mapsto 2^m$ is not definable in any existing system of bounded arithmetic.

5 Minimal function graphs

The *minimal function graph* semantics was described in [12] as denotational semantics, cf. [23, Chapter 9], and afterward used for termination analysis of functional programs without exponential size-

explosions in [11, Chapter 24.2] and [15]. In this section, we explain how minimal function graphs work, how they are defined inductively, and how they can be defined without inductive definitions.

To see how minimal function graphs work, consider the program $\mathbf{R}_{\mathsf{lcs}}$ in Example 1. Let us observe that the following reduction starting with the basic term $\mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\mathsf{b}(\mathsf{b}(\varepsilon)))$ is possible.

$$\begin{array}{ll} & \mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\mathsf{b}(\mathsf{b}(\varepsilon))) \\ \xrightarrow{\mathsf{i}}_{\mathbf{R}_{\mathsf{lcs}}} & \mathsf{max}(\mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\mathsf{b}(\varepsilon))),\mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\mathsf{b}(\varepsilon))) \\ \xrightarrow{\mathsf{i}}_{\mathbf{R}_{\mathsf{lcs}}} & \mathsf{max}(\mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\mathsf{b}(\varepsilon))),\mathsf{max}(\mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\varepsilon)),\mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\varepsilon))) \\ \xrightarrow{\mathsf{i}}_{\mathbf{R}_{\mathsf{lcs}}} & \mathsf{max}(\mathsf{max}(\mathsf{lcs}(\varepsilon,\mathsf{b}(\mathsf{b}(\varepsilon))),\mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\varepsilon))),\mathsf{max}(\mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\varepsilon)),\mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\varepsilon))) \end{array}$$

In the reduction, the term $t := \mathsf{lcs}(\mathsf{a}(\varepsilon),\mathsf{b}(\varepsilon))$ is duplicated, and hence costly re-computations potentially occur. For the same reason, there can be an exponential explosion in the size of the reduction tree rooted at $\mathsf{lcs}(\mathsf{a}(\mathsf{a}(\varepsilon)),\mathsf{b}(\mathsf{b}(\varepsilon)))$ that contains all the possible rewriting sequences starting with the basic term. A minimal function graph G, or *cache* in other words, is defined so that G stores pairs of a basic term and its normal form. Thus, once the term t is normalized to 0 (because the two strings a and b have no common subsequence), the pair $\langle t,0\rangle$ is stored in G and any other reduction of t can be simulated by replacing the occurrence of t with 0.

Given a program $\mathbf{R}$, a (variant of) minimal function graph G is defined as the least fixed point of the following operator Γ over $\mathscr{P}(\mathbf{B}(\mathbf{F})\times\mathbf{T}(\mathbf{C}))$, where $X \subseteq \mathbf{B}(\mathbf{F})\times\mathbf{T}(\mathbf{C})$.

$$\begin{array}{rcl} \langle t,s\rangle \in \Gamma(X) & :\Longleftrightarrow & \exists l\to r\in\mathbf{R}, \exists\theta:\mathbf{V}\to\mathbf{T}(\mathbf{C}), \exists\langle t_0,s_0\rangle,\ldots,\langle t_{\|r\|-1},s_{\|r\|-1}\rangle\in X \text{ s.t.} \\ & & t = l\theta \;\&\; s = \big((r\theta)[s_0/t_0]\cdots\big)[s_{\|r\|-1}/t_{\|r\|-1}] \end{array}$$

The operator Γ is monotone, i.e., $X\subseteq Y \Rightarrow \Gamma(X)\subseteq\Gamma(Y)$, and hence there exists the least fixed point of Γ. Suppose that $\mathbf{R}$ is quasi-reducible. On one side, the fixed-ness of G yields that $t \xrightarrow{\mathsf{i}}{}^{!}_{\mathbf{R}} s \Rightarrow \langle t,s\rangle\in G$. On the other side, since the set $\{\langle t,s\rangle \mid t\in\mathbf{B}(\mathbf{F}) \;\&\; t\xrightarrow{\mathsf{i}}{}^{!}_{\mathbf{R}} s\}$ is a fixed point of Γ, the least-ness of G yields that $\langle t,s\rangle\in G \Rightarrow t\xrightarrow{\mathsf{i}}{}^{!}_{\mathbf{R}} s$. Thus, to conclude that every closed basic term has an (innermost) $\mathbf{R}$-normal form, it suffices to show that, for every term $t\in\mathbf{B}(\mathbf{F})$, there exists a term s such that $\langle t,s\rangle\in G$. Now there are two important observations.

1. It suffices to show that, for every term $t\in\mathbf{B}(\mathbf{F})$, there exist a *subset* $G_t\subseteq G$ and a term s such that $\langle t,s\rangle\in G_t$. If $t=l\theta$ and $s=\big((r\theta)[s_0/t_0]\cdots\big)[s_{\|r\|-1}/t_{\|r\|-1}]$ as in the definition of Γ above and, for each $j<\|r\|$, $\langle t_j,s_j\rangle\in G_{t_j}$ holds for such a set $G_{t_j}\subseteq G$, then G_t can be simply defined as $G_t=\{\langle t,s\rangle\}\cup G_{t_0}\cup\cdots\cup G_{t_{\|r\|-1}}$.[1]
2. Additionally suppose that the program $\mathbf{R}$ reduces under an LPO $<_{\mathsf{lpo}}$. Then it turns out that the definition of Γ is equivalent to a form restricted in such a way that $t_j <_{\mathsf{lpo}} t$ for each $j<\|r\|$.[2]

For these reasons, the schema $(\forall t\in\mathbf{B}(\mathbf{F}))\big((\forall s<_{\mathsf{lpo}} t)\varphi(s)\to\varphi(t)\big)\to(\forall t\in\mathbf{B}(\mathbf{F}))\varphi(t)$ of transfinite induction along $<_{\mathsf{lpo}}$ will imply the termination of a quasi-reducible LPO-terminating program $\mathbf{R}$ in the sense above.

6 Formalizing LPO-termination proofs under PQIs in U^1_2

In this section, we show that, if $\mathbf{R}$ is a quasi-reducible $\mathrm{LPO}^{\mathrm{Poly}(0)}$-program, then an innermost $\mathbf{R}$-normal form of any closed basic term can be found in the system U^1_2 (Theorem 3).

[1] To be precise, in [11, 15], the *minimal function graph* was used to denote such a subset G_t for a given basic t.

[2] Namely, every function computed by an $<_{\mathsf{lpo}}$-reducing program is defined recursively along $<_{\mathsf{lpo}}$. Therefore, as a reviewer pointed out, in this case the minimal function graphs can be regarded as fixed-point semantics for recursive definitions of functions, cf. [19, Chapter 10].

Given a program $\mathbf{R}$ over a signature $\mathbf{F} = \mathbf{C} \cup \mathbf{D}$, we use the notation $V_{\mathbf{R}}$ to denote the finite set $\{x \in \mathbf{V} \mid x \text{ appears in some rule } \rho \in \mathbf{R}\}$ of variables. Let $\ulcorner \cdot \urcorner$ be an *efficient* binary encoding for $\mathbf{T}(\mathbf{F}, \mathbf{V}_{\mathbf{R}})$-terms. The efficiency means that:

(i) $t \mapsto \ulcorner t \urcorner$ is $\Sigma_0^{b,1}$-definable in U_2^1.

(ii) There exists a polynomial (term) $p(x)$ with a free variable x such that $|\ulcorner t \urcorner| \leq p(\|t\|)$ (provably) holds for any $t \in \mathbf{T}(\mathbf{F}, \mathbf{V}_{\mathbf{R}})$.

Without loss of generality, we can assume that:

(iii) $\|t\| \leq |\ulcorner t \urcorner|$.

(iv) $|\ulcorner s \urcorner| < |\ulcorner t \urcorner|$ if s is a proper subterm of t.

Such an encoding can be defined, for example, by representing terms as directed graphs not as trees.

Lemma 2. *The relation $<_{\mathsf{lpo}}$ is $\Sigma_0^{b,1}$-definable in U_2^1.*

Proof (Sketch). It suffices to show that, given two terms s and t, the relation "there exists a derivation tree according to the rules 1–3 (on page 36) that results in $s <_{\mathsf{lpo}} t$" is $\Sigma_0^{b,1}$-definable in U_2^1. Let T denote such a derivation tree resulting in $s <_{\mathsf{lpo}} t$. By induction according to the inductive definition of $<_{\mathsf{lpo}}$ it can be shown that the number of nodes in T is bounded by $\|s\| \cdot \|t\|$. Hence, by the assumption (ii) on the encoding $\ulcorner \cdot \urcorner$, the code $\ulcorner T \urcorner$ of T is polynomially bounded in $\|s\| \cdot \|t\|$ and thus in $\ulcorner s \urcorner \cdot \ulcorner t \urcorner$. On the other hand, by definition, the relation $s_0 <_{\mathsf{lpo}} t_0$ between two terms s_0 and t_0 is reduced to a tuple $s_j <_{\mathsf{lpo}} t_j$ $(j = 1, \ldots, k)$ of relations between some subterms $s_1, \ldots, s_k$ of s_0 and subterms $t_1, \ldots, t_k$ of t_0. Thanks to the assumption (iv) on the encoding $\ulcorner \cdot \urcorner$, $|\ulcorner s_j \urcorner| + |\ulcorner t_j \urcorner| < |\ulcorner s_0 \urcorner| + |\ulcorner t_0 \urcorner|$, i.e., $2^{|\ulcorner s_j \urcorner| + |\ulcorner t_j \urcorner|} \leq \lfloor \left(2^{|\ulcorner s_0 \urcorner| + |\ulcorner t_0 \urcorner|}\right)/2 \rfloor$, holds for any $j \in \{1, \ldots, k\}$. From these observations, it can be seen that the construction of the derivation tree T is performed in U_2^1, and hence the relation $s <_{\mathsf{lpo}} t$ is $\Sigma_0^{b,1}$-definable in U_2^1. □

As observed in [5], in which an optimal LPO-termination proof was described, every program $\mathbf{R}$ reducing under an LPO $<_{\mathsf{lpo}}$ already reduces under a finite restriction $<_\ell$ of $<_{\mathsf{lpo}}$ for some $\ell \in \mathbb{N}$ and every quantifier of the form $(Qs <_\ell t)$ can be regarded as a bounded one. Adopting the restriction, we introduce an even more restrictive relation $<_\ell$ $(\ell \in \mathbb{N})$ motivated by the following properties of PQIs.

Proposition 1. *Let $(\!| \cdot |\!)$ be a kind 0 PQI and $t \in \mathbf{B}(\mathbf{F})$. Then the following two properties hold.*

1. *$(\!|t|\!) \leq p(|\ulcorner t \urcorner|)$ holds for some polynomial p.*
2. *Suppose additionally that a program $\mathbf{R}$ admits the PQI $(\!| \cdot |\!)$ and that $t \xrightarrow{\mathrm{i}}{}^*_{\mathbf{R}} s$ holds. If $s \in \mathbf{T}(\mathbf{C})$, then $\|s\| \leq (\!|t|\!)$ holds. If $s = f(s_1, \ldots, s_k) \in \mathbf{B}(\mathbf{F})$, then $\|s_j\| \leq (\!|t|\!)$ holds for each $j \in \{1, \ldots, k\}$.*

Proof. PROPERTY 1. Let $t = g(t_1, \ldots, t_l)$. Since the PQI $(\!| \cdot |\!)$ is kind 0, one can find a constant d depending only on the set $\mathbf{C}$ of constructors and the PQI $(\!| \cdot |\!)$ such that $(\!|t_j|\!) \leq d \cdot \|t_j\|$ holds for any $j \in \{1, \ldots, l\}$. This yields a polynomial p such that $(\!|t|\!) \leq p(\|t\|)$ and thus $(\!|t|\!) \leq p(|\ulcorner t \urcorner|)$ holds by the assumption (iii) on the encoding $\ulcorner \cdot \urcorner$.

PROPERTY 2. In case $s \in \mathbf{T}(\mathbf{C})$, $\|s\| \leq (\!|s|\!) \leq (\!|t|\!)$ holds. In case $s = f(s_1, \ldots, s_k) \in \mathbf{B}(\mathbf{F})$, $\|s_j\| \leq (\!|s_j|\!) \leq (\!|s|\!) \leq (\!|t|\!)$ holds for each $j \in \{1, \ldots, k\}$. □

Definition 6 ($\mathbf{T}_\ell(\mathbf{C})$, $\mathbf{B}_\ell(\mathbf{F})$, $<_\ell$, $<_\ell^{\mathsf{lex}}$). Let $\mathbf{T}_\ell(\mathbf{C})$ denote a set $\{t \in \mathbf{T}(\mathbf{C}) \mid \|t\| \leq \ell\}$ of constructor terms and $\mathbf{B}_\ell(\mathbf{F})$ a set $\{f(t_1, \ldots, t_k) \in \mathbf{B}(\mathbf{F}) \mid \|t_1\|, \ldots, \|t_k\| \leq \ell\}$ of basic terms. Then we write $s <_\ell t$ if $s <_{\mathsf{lpo}} t$ and additionally $s \in \mathbf{T}_\ell(\mathbf{C}) \cup \mathbf{B}_\ell(\mathbf{F})$ hold. We use the notation $s <_\ell^{\langle i \rangle} t$ $(i = 1, 2, 3)$ accordingly. Moreover, we define a *lexicographic extension* $<_\ell^{\mathsf{lex}}$ of $<_\ell$ over $\mathbf{T}(\mathbf{C})$. For constructor terms $s_1, \ldots, s_k$, $t_1, \ldots, t_k$, we write $(s_1, \ldots, s_k) <_\ell^{\mathsf{lex}} (t_1, \ldots, t_k)$ if there exists an index $i \in \{1, \ldots, k\}$ such that $s_j = t_j$ for every $j < i$, $s_i <_\ell^{\langle 1 \rangle} t_i$, and $s_j \in \mathbf{T}_\ell(\mathbf{C})$ for every $j > i$.

Corollary 2 follows from the definitions of $<_\ell$ and $<_\ell^{\mathsf{lex}}$ and from $<_{\mathbf{F}}$-minimality of constructors.

Corollary 2. *For two basic terms $f(s_1,\dots,s_k), f(t_1,\dots,t_k) \in \mathbf{B}_\ell(\mathbf{F})$, $f(s_1,\dots,s_k) <_\ell^{\langle 3\rangle} f(t_1,\dots,t_k)$ holds if and only if $(s_1,\dots,s_k) <_\ell^{\mathsf{lex}} (t_1,\dots,t_k)$ holds.*

For most of interesting $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs including Example 1 and 2, interpreting polynomials consist of $+$, $\cdot$, $\max_{j=1}^k x_j$ together with additional constants. This motivates us to formalize PQIs limiting interpreting polynomial terms to those built up only from 0, S, $+$, $\cdot$ and max to make the formalization easier. Then the constraints (ii) and (iii) on PQIs follow from defining axioms for these function symbols.

Let us consider a reduction $t_0 \xrightarrow{\mathrm{i}}_{\mathbf{R}}^* t \xrightarrow{\mathrm{i}}_{\mathbf{R}}^* s$ under a program $\mathbf{R}$ admitting a kind 0 PQI $(\!|\cdot|\!)$, where $t_0, t \in \mathbf{B}(\mathbf{F})$ and $s \in \mathbf{T}(\mathbf{C}) \cup \mathbf{B}(\mathbf{F})$. If $s <_{\mathsf{lpo}} t$ for some LPO $<_{\mathsf{lpo}}$, then Proposition 1 yields a polynomial p such that $s <_{p(|\ulcorner t_0 \urcorner|)} t$ holds by Definition 6. Hence we can assume that ℓ is (the result of substituting t_0 for) a polynomial $p(|x|)$. More precisely, ℓ can be expressed by an $\mathbf{L}_{\mathsf{BA}}$-term built up from 0 and $|x|, |y|, |z|, \dots$ by S, $+$ and $\cdot$. By assumption, ℓ does not contain # nor $\lfloor \cdot/2 \rfloor$. Thus $\ell = \ell(x_1,\dots,x_k)$ denotes a polynomial with non-negative coefficients in $|x_1|,\dots,|x_k|$. Since ℓ contains no smash # in particular, $2^{p(\ell)}$ can be regarded as an $\mathbf{L}_{\mathsf{BA}}$-term for any polynomial $p(x)$. By the assumption (ii) on the encoding $\ulcorner \cdot \urcorner$, $|\ulcorner t \urcorner|$ is polynomially bounded in the size $\|t\|$ of t, and hence $\ulcorner t \urcorner \le 2^{p(\|t\|)}$ for some polynomial $p(x)$. Therefore any quantifier of the forms $(Qs <_\ell t)$, $(Qt \in \mathbf{T}_\ell(\mathbf{C}))$ and $(Qt \in \mathbf{B}_\ell(\mathbf{F}))$ can be treated as a bounded one.

We deduce the schema $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{B}_\ell(\mathbf{F}), <_\ell))$ of $<_\ell$-transfinite induction over $\mathbf{B}_\ell(\mathbf{F})$ for $\Sigma_1^{\mathrm{b},1}$-formulas (Lemma 5). Since the relation $f(s_1,\dots,s_k) <_\ell^{\langle 3\rangle} f(t_1,\dots,t_k)$ relies on the comparison $(s_1,\dots,s_k) <_\ell^{\mathsf{lex}} (t_1,\dots,t_k)$ by Corollary 2, we previously have to deduce the schema $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C})^k, <_\ell^{\mathsf{lex}}))$ of $<_\ell^{\mathsf{lex}}$-transfinite induction over k-tuples of $\mathbf{T}_\ell(\mathbf{C})$-terms (Lemma 4). We start with deducing the instance in the base case $k = 1$.

Lemma 3. *The following schema of $<_\ell$-transfinite induction over $\mathbf{T}_\ell(\mathbf{C})$ holds in U_2^1, where $\varphi \in \Sigma_1^{\mathrm{b},1}$.*

$$(\forall t \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall s <_\ell t)\varphi(s) \to \varphi(t)\big) \to (\forall t \in \mathbf{T}_\ell(\mathbf{C}))\varphi(t) \qquad (\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C}), <_\ell))$$

Proof. Reason in U_2^1. Suppose $(\forall t \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall s <_\ell t)\varphi(s) \to \varphi(t)\big)$ and let $t \in \mathbf{T}_\ell(\mathbf{C})$. We show that $\varphi(t)$ holds by $(\Sigma_1^{\mathrm{b},1}\text{-PIND})$ on $\ulcorner t \urcorner$. The case $\ulcorner t \urcorner = 0$ trivially holds. Suppose $\ulcorner t \urcorner > 0$ for induction step. By assumption, it suffices to show that $\varphi(s)$ holds for any $s <_\ell t$. Thus let $s <_\ell t$. Since $t \in \mathbf{T}_\ell(\mathbf{C})$, s is a proper subterm of t by Corollary 1 and $<_{\mathbf{F}}$-minimality of constructors. Thus, the assumption (iv) on the encoding $\ulcorner \cdot \urcorner$ yields $\ulcorner s \urcorner \le \lfloor \ulcorner t \urcorner/2 \rfloor$, and hence $\varphi(s)$ holds by induction hypothesis. □

Remark 1. In the proof of Lemma 3, we employed a bit-wise form of *course of values* induction $\varphi(0) \wedge \forall t\big(\forall s(\ulcorner s \urcorner \le \lfloor \ulcorner t \urcorner/2 \rfloor \to \varphi(s)) \to \varphi(t)\big) \to \forall t \varphi(t)$ for a $\Sigma_1^{\mathrm{b},1}$-formula $\varphi(x)$, which is not an instance of $(\Sigma_1^{\mathrm{b},1}\text{-PIND})$. Formally, one should apply $(\Sigma_1^{\mathrm{b},1}\text{-PIND})$ for the $\Sigma_1^{\mathrm{b},1}$-formula $\psi(x) \equiv \forall t\big(\ulcorner t \urcorner \le 2^{|x|} \to \varphi(t)\big)$ to deduce $(\forall t \in \mathbf{T}_\ell(\mathbf{C}))\,\varphi(t)$. To ease presentation, we will use similar informal arguments in the sequel.

Lemma 4. *The schema $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C}), <_\ell))$ can be extended to tuples of $\mathbf{T}_\ell(\mathbf{C})$-terms, i.e., the following schema holds in U_2^1, where $\varphi(\vec{t}) \equiv \varphi(t_1,\dots,t_k) \in \Sigma_1^{\mathrm{b},1}$.*

$$(\forall \vec{t} \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall \vec{s} <_\ell^{\mathsf{lex}} \vec{t})\varphi(\vec{s}) \to \varphi(\vec{t})\big) \to (\forall \vec{t} \in \mathbf{T}_\ell(\mathbf{C}))\varphi(\vec{t}) \qquad (\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C})^k, <_\ell^{\mathsf{lex}}))$$

Proof. We show that the schema $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C})^k, <_\ell^{\mathsf{lex}}))$ holds in U_2^1 by (meta) induction on $k \ge 1$. In case $k = 1$, the schema is an instance of $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C}), <_\ell))$. Suppose that $k > 1$ and $(\mathsf{TI}_{\Sigma_1^{\mathrm{b},1}}(\mathbf{T}_\ell(\mathbf{C})^{k-1}, <_\ell^{\mathsf{lex}}))$ holds by induction hypothesis. Assume that

$$(\forall t_1,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall (s_1,\dots,s_k) <_\ell^{\mathsf{lex}} (t_1,\dots,t_k))\varphi(s_1,\dots,s_k) \to \varphi(t_1,\dots,t_k)\big) \qquad (1)$$

holds for some $\Sigma^{\mathrm{b},1}_1$-formula $\varphi(t_1,\dots,t_k)$. Let $\varphi_{<^{\mathsf{lex}}_\ell}(t,t_2,\dots,t_k)$, $\psi(t)$ and $\psi_{<_\ell}(t)$ denote $\Sigma^{\mathrm{b},1}_1$-formulas specified as follows.

$$\begin{array}{rcl}\varphi_{<^{\mathsf{lex}}_\ell}(t,t_2,\dots,t_k) & :\equiv & t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}) \wedge \big(\forall(s_2,\dots,s_k) <^{\mathsf{lex}}_\ell (t_2,\dots,t_k)\big)\,\varphi(t,s_2,\dots,s_k);\\ \psi(t) & :\equiv & (\forall t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\,\varphi(t,t_2,\dots,t_k);\\ \psi_{<_\ell}(t) & :\equiv & t \in \mathbf{T}_\ell(\mathbf{C}) \wedge (\forall s <_\ell t)\psi(s).\end{array}$$

Note, in particular, that $\psi(t)$ is still a $\Sigma^{\mathrm{b},1}_1$-formula since every quantifier of the form $(\forall s \in \mathbf{T}_\ell(\mathbf{C}))$ can be regarded as a bounded one under which the class $\Sigma^{\mathrm{b},1}_1$ is closed. One can see that $\varphi_{<^{\mathsf{lex}}_\ell}(t,t_2,\dots,t_k)$ and $\psi_{<_\ell}(t)$ imply $t,t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C})$ and $\big(\forall(s,s_2,\dots,s_k) <^{\mathsf{lex}}_\ell (t,t_2,\dots,t_k)\big)\varphi(s,s_2,\dots,s_k)$. Hence, by the assumption (1), $\psi_{<_\ell}(t)$ implies $(\forall t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\big(\varphi_{<^{\mathsf{lex}}_\ell}(t,t_2,\dots,t_k) \to \varphi(t,t_2,\dots,t_k)\big)$, which denotes

$$(\forall t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall(s_2,\dots,s_k) <^{\mathsf{lex}}_\ell (t_2,\dots,t_k))\varphi(t,s_1,\dots,s_k) \to \varphi(t,t_2,\dots,t_k)\big).$$

This together with $(\mathsf{TI}_{\Sigma^{\mathrm{b},1}_1}(\mathbf{T}_\ell(\mathbf{C})^{k-1}, <^{\mathsf{lex}}_\ell))$ yields $(\forall t_2,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\,\varphi(t,t_2,\dots,t_k)$, denoting $\psi(t)$. This means that $(\forall t \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall s <_\ell t)\psi(s) \to \psi(t)\big)$ holds. Since $\psi(t) \in \Sigma^{\mathrm{b},1}_1$ as noted above, this together with $(\mathsf{TI}_{\Sigma^{\mathrm{b},1}_1}(\mathbf{T}_\ell(\mathbf{C}), <_\ell))$ yields $(\forall t \in \mathbf{T}_\ell(\mathbf{C}))\psi(t)$ and thus $(\forall t_1,\dots,t_k \in \mathbf{T}_\ell(\mathbf{C}))\varphi(t_1,\dots,t_k)$ holds. □

Lemma 5. *Let* $\mathbf{F} = \mathbf{C} \cup \mathbf{D}$. *The* $<_\ell$*-transfinite induction over* $\mathbf{B}_\ell(\mathbf{F})$ *holds in* U^1_2, *where* $\varphi \in \Sigma^{\mathrm{b},1}_1$.

$$(\forall t \in \mathbf{B}_\ell(\mathbf{F}))\big((\forall s \in \mathbf{B}_\ell(\mathbf{F}))(s <_\ell t \to \varphi(s)) \to \varphi(t)\big) \to (\forall t \in \mathbf{B}_\ell(\mathbf{F}))\varphi(t) \qquad (\mathsf{TI}_{\Sigma^{\mathrm{b},1}_1}(\mathbf{B}_\ell(\mathbf{F}), <_\ell))$$

Given a precedence $<_{\mathbf{F}}$ on the finite signature $\mathbf{F}$, let $\mathsf{rk} : \mathbf{F} \to \mathbb{N}$ denote the *rank*, a finite function compatible with $<_{\mathbf{F}}$: $\mathsf{rk}(f) < \mathsf{rk}(g) \Leftrightarrow f <_{\mathbf{F}} g$.

Proof. Reason in U^1_2. Assume the premise of $(\mathsf{TI}_{\Sigma^{\mathrm{b},1}_1}(\mathbf{B}_\ell(\mathbf{F}), <_\ell))$:

$$(\forall t \in \mathbf{B}_\ell(\mathbf{F}))\big((\forall s \in \mathbf{B}_\ell(\mathbf{F}))(s <_\ell t \to \varphi(s)) \to \varphi(t)\big) \tag{2}$$

Let $g \in \mathbf{D}$. We show that $(\forall t_1,\dots,t_l \in \mathbf{T}_\ell(\mathbf{C}))\varphi(g(t_1,\dots,t_l))$ holds by $(\Sigma^{\mathrm{b},1}_1\text{-}\mathsf{PIND})$ on $2^{\mathsf{rk}(g)}$, or in other words by finitary induction on $\mathsf{rk}(g)$. Let $t_1,\dots,t_l \in \mathbf{T}_\ell(\mathbf{C})$ and $t := g(t_1,\dots,t_l)$. By the assumption (2), it suffices to show that $\varphi(s)$ holds for any $s \in \mathbf{B}_\ell(\mathbf{F})$ such that $s <_\ell t$. Thus, let $s \in \mathbf{B}_\ell(\mathbf{F})$ and $s <_\ell t$.

Case. $s <^{\langle 1\rangle}_\ell t$: In this case $s \leqslant_\ell t_i$ for some $i \in \{1,\dots,l\}$. Since $t_i \in \mathbf{T}_\ell(\mathbf{C})$, $s \in \mathbf{T}_\ell(\mathbf{C})$ as well by Corollary 1, and hence this case is excluded.

Case. $s := f(s_1,\dots,s_k) <^{\langle 2\rangle}_\ell t$: In this case, $f <_{\mathbf{F}} g$ and hence $\mathsf{rk}(f) < \mathsf{rk}(g)$. This allows us to reason as $2^{\mathsf{rk}(g)} \leq 2^{\mathsf{rk}(f)-1} = \lfloor 2^{\mathsf{rk}(f)}/2 \rfloor$. Thus the induction hypothesis yields $\varphi(s)$.

Case. $s := g(s_1,\dots,s_l) <^{\langle 3\rangle}_\ell t$: We show that the following condition holds.

$$(\forall v_1,\dots,v_l \in \mathbf{T}_\ell(\mathbf{C}))\big((\forall(u_1,\dots,u_l) <^{\mathsf{lex}}_\ell (v_1,\dots,v_l))\varphi(g(u_1,\dots,u_l)) \to \varphi(g(v_1,\dots,v_l))\big) \tag{3}$$

Let $v_1,\dots,v_l \in \mathbf{T}_\ell(\mathbf{C})$. By Corollary 2, the premise $(\forall(u_1,\dots,u_l) <^{\mathsf{lex}}_\ell (v_1,\dots,v_l))\varphi(g(u_1,\dots,u_l)$ of (3) yields $\big(\forall s' <^{\langle 3\rangle}_\ell g(v_1,\dots,v_l)\big)\,\varphi(s')$. On the other side, the previous two cases yield $(\forall s' \in \mathbf{B}_\ell(\mathbf{F}))\big(s' <^{\langle i\rangle}_\ell g(v_1,\dots,v_l) \to \varphi(s')\big)$ $(i = 1,2)$ and hence $(\forall s' \in \mathbf{B}_\ell(\mathbf{F}))\big(s' <_\ell g(v_1,\dots,v_l) \to \varphi(s')\big)$ holds. Therefore $\varphi(g(v_1,\dots,v_l))$ holds by the assumption (2), yielding the statement (3). Since (3) is the premise of an instance of the schema $(\mathsf{TI}_{\Sigma^{\mathrm{b},1}_1}(\mathbf{T}_\ell(\mathbf{C})^l, <^{\mathsf{lex}}_\ell))$, Lemma 4 yields $(\forall v_1,\dots,v_l \in \mathbf{T}_\ell(\mathbf{C}))\varphi(g(v_1,\dots,v_l))$, and thus $\varphi(g(s_1,\dots,s_l))$ holds in particular. □

To derive, from $(\mathsf{TI}_{\Sigma_1^{b,1}}(\mathbf{B}_\ell(\mathbf{F}),<_\ell))$, the existence of a minimal function graph under an LPO-terminating program, we need the following technical lemma.

Lemma 6. (in U_2^1) *Let $(\!|\cdot|\!)$ be a kind 0 PQI for a signature $\mathbf{F}=\mathbf{C}\cup\mathbf{D}$, $t\in\mathbf{B}(\mathbf{F})$, $s\in\mathbf{T}(\mathbf{F})$ and $<_{\mathsf{lpo}}$ an LPO induced by a precedence $<_\mathbf{F}$. If $s<_{\mathsf{lpo}}t$ and $(\!|s|\!)\le(\!|t|\!)\le\ell$, then, for any basic subterm t' of s and for any $s'\in\mathbf{T}(\mathbf{C})$ such that $(\!|s'|\!)\le(\!|t'|\!)$, $v<_\ell t$ holds for any basic subterm v of $s[s'/t']$.*

Proof. By $<_\mathbf{F}$-minimality of constructors, $s'<_{\mathsf{lpo}}t'$ holds. Hence $s[s'/t']<_{\mathsf{lpo}}t$ from the assumption $s<_{\mathsf{lpo}}t$. This yields $v<_{\mathsf{lpo}}t$ by the definition of LPOs. Write $v=f(v_1,\ldots,v_k)$ for some $f\in\mathbf{D}$ and $v_1,\ldots,v_k\in\mathbf{T}(\mathbf{C})$. Let $i\in\{1,\ldots,k\}$. Then $\|v_i\|\le(\!|v_i|\!)\le(\!|v|\!)\le(\!|s[s'/t']|\!)\le(\!|t|\!)$. The last inequality follows from the monotonicity (ii) of the PQI $(\!|\cdot|\!)$. This yields $\|v_i\|\le\ell$ and hence $v<_\ell t$. □

Theorem 3. (in U_2^1) *Suppose that $\mathbf{R}$ is a quasi-reducible $\mathrm{LPO}^{Poly(0)}$-program. Then, for any basic term t, there exists a minimal function graph G (in the sense of Section 5) such that that $\langle t,s\rangle\in G$ holds for an $\mathbf{R}$-normal form s of t.*

Proof. Suppose that $\mathbf{R}$ is a quasi-reducible $\mathrm{LPO}^{\mathrm{Poly}(0)}$-program witnessed by an LPO $<_{\mathsf{lpo}}$ and a kind 0 PQI $(\!|\cdot|\!)$ and that $<_\ell$ is a finite restriction of $<_{\mathsf{lpo}}$. Let $\psi_\ell(x,y,X)$ denote a $\Sigma_0^{b,1}$-formula with no free variables other than x, y and X expressing that $X\subseteq\mathbf{B}_\ell(\mathbf{F})\times\mathbf{T}_\ell(\mathbf{C})$ is a set of pairs of terms such that $\langle x,y\rangle\in X$, and, for any $\langle t,s\rangle\in X$, $(\!|s|\!)\le(\!|t|\!)\le\ell$ and $\exists l\to r\in\mathbf{R}$, $\exists\theta:V_\mathbf{R}\to\mathbf{T}_\ell(\mathbf{C})$ s.t. $t=l\theta$ and one of the following cases holds.

1. $s=r\theta\in\mathbf{T}_\ell(\mathbf{C})$.
2. $\exists\langle\langle t_j,s_j\rangle\in X\mid j<\|r\|\rangle$ s.t. $s=\big((r\theta)[s_0/t_0]\cdots\big)[s_{\|r\|-1}/t_{\|r\|-1}]$, where $s'[u/v]$ is identical if no v occurs in s'.

Note that, since $V_\mathbf{R}$ is a finite set of variables, $\exists\theta:V_\mathbf{R}\to\mathbf{T}_\ell(\mathbf{C})$ can be regarded as a (first order) bounded quantifier. By Proposition 1.1, we can find a polynomial term $p(x)$ such that $(\!|t|\!)\le p(|\ulcorner t\urcorner|)$ holds for any $t\in\mathbf{B}(\mathbf{F})$. The rest of the proof is devoted to deduce $(\forall t\in\mathbf{B}(\mathbf{F}))(\exists s\in\mathbf{T}_\ell(\mathbf{C}))\exists G\,\psi_{p(|\ulcorner t\urcorner|)}(t,s,G)$ for such a bounding polynomial p. Fix an input basic term $t_0\in\mathbf{B}(\mathbf{F})$ and let $\varphi_\ell(t)$ denote the $\Sigma_1^{b,1}$-formula $(\exists s\in\mathbf{T}_\ell(\mathbf{C}))\exists G\,\psi_\ell(t,s,G)$, where $\ell=p(|\ulcorner t_0\urcorner|)$. Since $t_0\in\mathbf{B}_\ell(\mathbf{F})$, it suffices to deduce $(\forall t\in\mathbf{B}_\ell(\mathbf{F}))\varphi_\ell(t)$. By Lemma 5, this follows from $(\forall t\in\mathbf{B}_\ell(\mathbf{F}))\big((\forall s\in\mathbf{B}_\ell(\mathbf{F}))(s<_\ell t\to\varphi_\ell(s))\to\varphi_\ell(t)\big)$, which is the premise of an instance of $(\mathsf{TI}_{\Sigma_1^{b,1}}(\mathbf{B}_\ell(\mathbf{F}),<_\ell))$. Thus let $t\in\mathbf{B}_\ell(\mathbf{F})$ and assume the condition

$$(\forall s\in\mathbf{B}_\ell(\mathbf{F}))(s<_\ell t\to\varphi_\ell(s)). \tag{4}$$

Since $\mathbf{R}$ is quasi-reducible, there exist a rule $l\to r\in\mathbf{R}$ and a substitution $\theta:V_\mathbf{R}\to\mathbf{T}_\ell(\mathbf{C})$ such that $t=l\theta$. The remaining argument splits into two cases depending on the shape of $r\theta$.

CASE 1. $r\theta\in\mathbf{T}_\ell(\mathbf{C})$: In this case $\psi_\ell(t,r\theta,G)$ holds for the singleton $G:=\{\langle t,r\theta\rangle\}$.

CASE 2. $r\theta\notin\mathbf{T}_\ell(\mathbf{C})$: In this case there exists a basic subterm v_0 of $r\theta$. Fix a term $u_0\in\mathbf{T}_\ell(\mathbf{C})$ such that $(\!|u_0|\!)\le(\!|v_0|\!)$. We show the following claim by finitary induction on $m<\|r\|$.

Claim 1. *There exists a sequence $\big\langle\langle t_j,s_j,G_j\rangle\mid j\le m\big\rangle$ of triplets such that, for each $j\le m$,* (i) *$t_j<_\ell t$,* (ii) *$\psi_\ell(t_j,s_j,G_j)$ holds, and* (iii) *$\big((r\theta)[s_0/t_0]\cdots\big)[s_j/t_j]$ is not identical to $\big((r\theta)[s_0/t_0]\cdots\big)[s_{j-1}/t_{j-1}]$ as long as $\big((r\theta)[s_0/t_0]\cdots\big)[s_{j-1}/t_{j-1}]$ has a basic subterm.*

In the base case $m=0$, let t_0 be an arbitrary basic subterm of $r\theta$. Then, since $(\!|r\theta|\!)\le(\!|l\theta|\!)$, $t_0<_\ell t$ follows from the definition of LPOs. Hence, by the assumption (4), there exist a term $s_0\in\mathbf{T}_\ell(\mathbf{C})$ and a set G_0 such that $\psi_\ell(t_0,s_0,G_0)$ holds. Clearly, $(r\theta)[s_0/t_0]$ is not identical to $r\theta$. For induction step,

suppose that there exists a sequence $\langle\langle t_j, s_j, G_j\rangle \mid j \le m\rangle$ fulfilling the conditions (i)–(iii) in the claim. In case that $\big((r\theta)[s_0/t_0]\cdots\big)[s_m/t_m]$ has no basic subterm, let $(t_{m+1}, s_{m+1}) = (v_0, u_0)$. Otherwise, let t_{m+1} be an arbitrary basic subterm. Then $t_{m+1} <_\ell t$ holds by Lemma 6. Hence, as in the base case, the assumption (4) yields a term $s_{m+1} \in \mathbf{T}_\ell(\mathbf{C})$ and a set G_{m+1} such that $\psi_\ell(t_{m+1}, s_{m+1}, G_{m+1})$ holds. By the choice of t_{m+1}, $\big((r\theta)[s_0/t_0]\cdots\big)[s_{m+1}/t_{m+1}]$ is not identical to $\big((r\theta)[s_0/t_0]\cdots\big)[s_m/t_m]$.

Now let $s := \big((r\theta)[s_0/t_0]\cdots\big)[s_{\|r\|-1}/t_{\|r\|-1}]$ for a sequence $\langle\langle t_j, s_j, G_j\rangle \mid j < \|r\|\rangle$ witnessing the claim in case $m = \|r\| - 1$. Then $s \in \mathbf{T}_\ell(\mathbf{C})$ since $|\{f \in \mathbf{D} \mid f \text{ appears in } \big((r\theta)[s_0/t_0]\cdots\big)[s_j/t_j]\}| \le \|r\| - (j+1)$ holds for each $j < \|r\|$ by the condition (iii) in the claim. Defining a set G by $G = \{\langle t, s\rangle\} \cup \left(\bigcup_{j<\|r\|} G_j\right)$ now allows us to conclude $\psi_\ell(t, s, G)$. □

7 Application

In the last section, to convince readers that the formalization of termination proofs described in Theorem 3 for $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs is optimal, we show that the formalization yields an alternative proof of Theorem 1, i.e., that $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs can only compute polynomial-space computable functions.

The next lemma ensures that the set G constructed in Theorem 3 is indeed a minimal function graph.

Lemma 7. *Suppose that* $\mathbf{R}$ *is a quasi-reducible* $\mathrm{LPO}^{\mathrm{Poly}(0)}$*-program. Let* $\psi_\ell(x, y, X)$ *denote the* $\Sigma_0^{\mathrm{b},1}$*-formula defined in the proof of Theorem 3. Then, for any* $t \in \mathbf{B}(\mathbf{F})$ *and for any* $t \in \mathbf{T}(\mathbf{C})$, $t \xrightarrow{\mathrm{i}}{}^{!}_{\mathbf{R}} s$ *if and only if* $\exists G\, \psi_{p(|\ulcorner t \urcorner|)}(t, s, G)$ *holds under the standard semantics.*

Proof. Let $\mathbf{R}$ reduce under an LPO $<_{\mathsf{lpo}}$. For the "if" direction, it can be shown that $(\forall t \in \mathbf{B}(\mathbf{F}))(\forall s \in \mathbf{T}(\mathbf{C}))\big(\exists G\, \psi_{p(|\ulcorner t \urcorner|)}(t, s, G) \Rightarrow t \xrightarrow{\mathrm{i}}{}^{!}_{\mathbf{R}} s\big)$ holds by (external) transfinite induction along $<_{\mathsf{lpo}}$. For the "only if" direction, it can be shown that $(\forall t \in \mathbf{B}(\mathbf{F}))(\forall s \in \mathbf{T}(\mathbf{C}))\big(t \xrightarrow{\mathrm{i}}{}^{m}_{\mathbf{R}} s \Rightarrow \exists G\, \psi_{p(|\ulcorner t \urcorner|)}(t, s, G)\big)$ holds by induction on m, where $\xrightarrow{\mathrm{i}}{}^{m}_{\mathbf{R}}$ denotes the m-fold iteration of $\xrightarrow{\mathrm{i}}_{\mathbf{R}}$. □

Now Theorem 3 and Lemma 7 yield an alternative proof of (a variant of) Theorem 1.

Corollary 3. *Every function computed by a quasi-reducible* $\mathrm{LPO}^{\mathrm{Poly}(0)}$*-program is computable in polynomial space.*

Proof. By Theorem 3, U_2^1 proves the formula

$$\mathsf{QR}(\mathbf{R}) \wedge \mathsf{LPO}(\mathbf{R}, <_{\mathsf{lpo}}) \wedge \mathsf{PQI}(\mathbf{R}, (\!|\cdot|\!)) \to (\forall t \in \mathbf{B}(\mathbf{F}))(\exists s \in \mathbf{T}_{p(|\ulcorner t \urcorner|)}(\mathbf{C}))\exists G\, \psi_{p(|\ulcorner t \urcorner|)}(t, s, G),$$

where $\mathsf{QR}(\mathbf{R})$, $\mathsf{LPO}(\mathbf{R}, <_{\mathsf{lpo}})$ and $\mathsf{PQI}(\mathbf{R}, (\!|\cdot|\!))$ respectively express that any $\mathbf{B}(\mathbf{F})$-term is reducible, $\mathbf{R}$ reduces under $<_{\mathsf{lpo}}$, and $(\forall (l \to r) \in \mathbf{R})(\forall \theta : V_{\mathbf{R}} \to \mathbf{T}(\mathbf{C}))(\!|r\theta|\!) \le (\!|l\theta|\!)$. By Lemma 2, $\mathsf{LPO}(\mathbf{R}, <_{\mathsf{lpo}})$ can be expressed with a $\Sigma_0^{\mathrm{b},1}$-formula, but neither $\mathsf{QR}(\mathbf{R})$ nor $\mathsf{PQI}(\mathbf{R}, (\!|\cdot|\!))$ is literally expressible with a bounded formula. Nonetheless, the proof can be easily modified to a proof of the statement

$$(\forall t \in \mathbf{B}(\mathbf{F}))(\exists s \in \mathbf{T}_\ell(\mathbf{C}))\big(\mathsf{QR}_\ell(\mathbf{R}) \wedge \mathsf{LPO}(\mathbf{R}, <_{\mathsf{lpo}}) \wedge \mathsf{PQI}_\ell(\mathbf{R}, (\!|\cdot|\!)) \to \exists G\, \psi_\ell(t, s, G)\big),$$

where $\ell = p(|\ulcorner t \urcorner|)$, and $\mathsf{QR}_\ell(\mathbf{R})$ and $\mathsf{PQI}_\ell(\mathbf{R}, (\!|\cdot|\!))$ respectively express that any $\mathbf{B}_\ell(\mathbf{F})$-term is reducible, and $(\forall (l \to r) \in \mathbf{R})(\forall \theta : V_{\mathbf{R}} \to \mathbf{T}_\ell(\mathbf{C}))(\!|r\theta|\!) \le (\!|l\theta|\!)$. Both $\mathsf{QR}_\ell(\mathbf{R})$ and $\mathsf{PQI}_\ell(\mathbf{R}, (\!|\cdot|\!))$ can be regarded as $\Sigma_0^{\mathrm{b},1}$-formulas, and hence the formula $\varphi_\ell(t, s) :\equiv \mathsf{QR}_\ell(\mathbf{R}) \wedge \mathsf{LPO}(\mathbf{R}, <_{\mathsf{lpo}}) \wedge \mathsf{PQI}_\ell(\mathbf{R}, (\!|\cdot|\!)) \to \exists G\, \psi_\ell(t, s, G)$ lies in $\Sigma_1^{\mathrm{b},1}$.

Now suppose that a function $[\![\mathsf{f}]\!] : \mathbf{T}(\mathbf{C})^k \to \mathbf{T}(\mathbf{C})$ is computed by a quasi-reducible $\mathrm{LPO}^{\mathrm{Poly}(0)}$-program $\mathbf{R}$ for some k-ary function symbol $\mathsf{f} \in \mathbf{D}$. Then Lemma 1 yields a polynomial-space computable

function $f:\mathbb{N}^k\to\mathbb{N}$ such that $\varphi_{p(|\ulcorner \mathsf{f}(t_1,\ldots,t_k)\urcorner|)}\big(\mathsf{f}(t_1,\ldots,t_k),\underline{f(\ulcorner t_1\urcorner,\ldots,\ulcorner t_k\urcorner)}\big)$ holds for any $t_1,\ldots,t_k\in\mathbf{T}(\mathbf{C})$ under the standard semantics. Hence, by assumption, $\psi_{p(|\ulcorner \mathsf{f}(t_1,\ldots,t_k)\urcorner|)}\big(\mathsf{f}(t_1,\ldots,t_k),\underline{f(\ulcorner t_1\urcorner,\ldots,\ulcorner t_k\urcorner)},G\big)$ holds for some set $G\subseteq\mathbf{B}(\mathbf{F})\times\mathbf{T}(\mathbf{C})$. By Lemma 7, this means the correspondence $\underline{[\![\mathsf{f}]\!](t_1,\ldots,t_k)=s}\Leftrightarrow f(\ulcorner t_1\urcorner,\ldots,\ulcorner t_k\urcorner)=\ulcorner s\urcorner$. Therefore, $\ulcorner[\![\mathsf{f}]\!](t_1,\ldots,t_k)\urcorner$ can be computed with space bounded by a polynomial in $|\ulcorner t_1\urcorner|,\ldots,|\ulcorner t_k\urcorner|$ and thus bounded by a polynomial in $\|t_1\|,\ldots,\|t_k\|$. □

8 Conclusion

This work is concerned with optimal termination proofs for functional programs in the hope of establishing logical foundations of computational resource analysis. Optimal termination proofs were limited for programs that compute functions lying in complexity classes closed under exponentiation. In this paper, employing the notion of minimal function graph, we showed that termination proofs under $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs can be optimally formalized in the second order system U^1_2 of bounded arithmetic that is complete for polynomial-space computable functions, lifting the limitation. The crucial idea is that inductive definitions of minimal function graphs under $\mathrm{LPO}^{\mathrm{Poly}(0)}$-programs can be approximated with transfinite induction along LPOs. As a small consequence, compared to the original result, Theorem 1, when we say "a program $\mathbf{R}$ computes a function", the quasi-reducibility of $\mathbf{R}$ is explicitly needed to enable the formalization.

Finally, let us call a program $\mathbf{R}$ an *MPO*$^{\mathit{Poly}(0)}$ one if $\mathbf{R}$ reduces under an MPO (with product status only) and $\mathbf{R}$ admits a kind 0 PQI. In [4, Theorem 42], Theorem 1 is refined so that a function can be computed by an $\mathrm{MPO}^{\mathrm{Poly}(0)}$-program if and only if it is computable in polynomial time. The program $\mathbf{R}_{\mathsf{lcs}}$ described in Example 1 is an example of $\mathrm{MPO}^{\mathrm{Poly}(0)}$-programs, and hence the length of the longest common subsequences is computable even in polynomial time. By Theorem 2.1, it is quite natural to expect that minimal function graphs under $\mathrm{MPO}^{\mathrm{Poly}(0)}$-programs can be constructed in the first order system S^1_2. However, we then somehow have to adopt the formula $\varphi_\ell(t,s)\equiv\mathsf{QR}_\ell(\mathbf{R})\wedge\mathsf{LPO}(\mathbf{R},<_{\mathsf{lpo}})\wedge\mathsf{PQI}_\ell(\mathbf{R},(\!|\cdot|\!))\to\exists G\ \psi_\ell(t,s,G)$ (in the proof of Corollary 3) to a Σ^{b}_1-formula, which is clearly more involved than the present case.

References

[1] A. Beckmann & S.R. Buss (2014): *Improved Witnessing and Local Improvement Principles for Second-order Bounded Arithmetic*. *ACM Transactions on Computational Logic* 15(1), p. 2, doi:10.1145/2559950.

[2] G. Bonfante, A. Cichon, J.-Y. Marion & H. Touzet (2001): *Algorithms with Polynomial Interpretation Termination Proof*. *Journal of Functional Programming* 11(1), pp. 33–53, doi:10.1017/S0956796800003877.

[3] G. Bonfante, J.-Y. Marion & J.-Y. Moyen (2001): *On Lexicographic Termination Ordering with Space Bound Certifications*. In: *Perspectives of System Informatics, Lecture Notes in Computer Science* 2244, pp. 482–493, doi:10.1007/3-540-45575-2_46.

[4] G. Bonfante, J.-Y. Marion & J.-Y. Moyen (2011): *Quasi-interpretations A Way to Control Resources*. *Theoretical Computer Science* 412(25), pp. 2776–2796, doi:10.1016/j.tcs.2011.02.007.

[5] W. Buchholz (1995): *Proof-theoretic Analysis of Termination Proofs*. *Annals of Pure and Applied Logic* 75(1–2), pp. 57–65, doi:10.1016/0168-0072(94)00056-9.

[6] S.R. Buss (1986): *Bounded Arithmetic*. Bibliopolis, Napoli.

[7] S.R. Buss (1998): *First-Order Proof Theory of Arithmetic*. In S.R. Buss, editor: *Handbook of Proof Theory*, North Holland, Amsterdam, pp. 79–147, doi:10.1016/S0049-237X(98)80017-7.

[8] N. Dershowitz (1982): *Orderings for Term-Rewriting Systems*. *Theoretical Computer Science* 17, pp. 279–301, doi:10.1016/0304-3975(82)90026-3.

[9] N. Eguchi (2010): *A Term-rewriting Characterization of PSPACE*. In T. Arai, C.T. Chong, R. Downey, J. Brendle, Q. Feng, H. Kikyo & H. Ono, editors: *Proceedings of the 10th Asian Logic Conference 2008*, World Scientific, pp. 93–112, doi:10.1142/9789814293020_0004.

[10] D. Hofbauer (1990): *Termination Proofs by Multiset Path Orderings Imply Primitive Recursive Derivation Lengths*. In: *Proceedings of the 2nd International Conference on Algebraic and Logic Programming, Lecture Notes in Computer Science* 463, pp. 347–358, doi:10.1007/3-540-53162-9_50.

[11] N.D. Jones (1997): *Computability and Complexity - from a Programming Perspective*. Foundations of Computing Series, MIT Press, doi:10.1007/978-94-010-0413-8_4.

[12] N.D. Jones & A. Mycroft (1986): *Data Flow Analysis of Applicative Programs Using Minimal Function Graphs*. In: *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pp. 296–306, doi:10.1145/512644.512672.

[13] S. Kamin & J.-J. Lévy (1980): *Two Generalizations of the Recursive Path Ordering*. Unpublished manuscript, University of Illinois.

[14] D. Leivant & J.-Y. Marion (1995): *Ramified Recurrence and Computational Complexity II: Substitution and Poly-space*. *Lecture Notes in Computer Science* 933, pp. 486–500, doi:10.1007/BFb0022277.

[15] J.-Y. Marion (2003): *Analysing the Implicit Complexity of Programs*. *Information and Computation* 183(1), pp. 2–18, doi:10.1016/S0890-5401(03)00011-7.

[16] I. Oitavem (2001): *Implicit Characterizations of Pspace*. In: *Proof Theory in Computer Science, Lecture Notes in Computer Science* 2183, Springer, pp. 170–190, doi:10.1007/3-540-45504-3_11.

[17] I. Oitavem (2002): *A Term Rewriting Characterization of the Functions Computable in Polynomial Space*. *Archive for Mathematical Logic* 41(1), pp. 35–47, doi:10.1007/s001530200002.

[18] W.J. Savitch (1970): *Relationships Between Nondeterministic and Deterministic Tape Complexities*. *Journal of Computer and System Sciences* 4(2), pp. 177–192, doi:10.1016/S0022-0000(70)80006-X.

[19] K. Slonneger & B.L. Kurtz (1995): *Formal Syntax and Semantics of Programming Languages - A Laboratory Based Approach*. Addison-Wesley.

[20] Terese (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

[21] D.B. Thompson (1972): *Subrecursiveness: Machine-Independent Notions of Computability in Restricted Time and Storage*. *Mathematical Systems Theory* 6(1), pp. 3–15, doi:10.1007/BF01706069.

[22] A. Weiermann (1995): *Termination Proofs for Term Rewriting Systems by Lexicographic Path Orderings Imply Multiply Recursive Derivation Lengths*. *Theoretical Computer Science* 139(1&2), pp. 355–362, doi:10.1016/0304-3975(94)00135-6.

[23] G. Winskel (1993): *The Formal Semantics of Programming Languages - An Introduction*. Foundations of Computing Series, MIT Press.

*-Continuous Kleene ω-Algebras for Energy Problems*

Zoltán Ésik
University of Szeged, Hungary

Uli Fahrenberg
Axel Legay
Inria Rennes, France

Energy problems are important in the formal analysis of embedded or autonomous systems. Using recent results on *-continuous Kleene ω-algebras, we show here that energy problems can be solved by algebraic manipulations on the transition matrix of energy automata. To this end, we prove general results about certain classes of finitely additive functions on complete lattices which should be of a more general interest.

1 Introduction

Energy problems are concerned with the question whether a given system admits infinite schedules during which (1) certain tasks can be repeatedly accomplished and (2) the system never runs out of energy (or other specified resources). These are important in areas such as embedded systems or autonomous systems and, starting with [4], have attracted some attention in recent years, for example in [3, 5–8, 16, 19, 23, 24].

With the purpose of generalizing some of the above approaches, we have in [12,17] introduced *energy automata*. These are finite automata whose transitions are labeled with *energy functions* which specify how energy values change from one system state to another. Using the theory of semiring-weighted automata [9], we have shown in [12] that energy problems in such automata can be solved in a simple static way which only involves manipulations of energy functions.

In order to put the work of [12] on a more solid theoretical footing and with an eye to future generalizations, we have recently introduced a new algebraic structure of **-continuous Kleene ω-algebras* [10] (see also [11] for the long version). We show here that energy functions form such a *-continuous Kleene ω-algebra. Using the fact, proven in [10], that for automata with transition weights in *-continuous Kleene ω-algebras, reachability and Büchi acceptance can be computed by algebraic manipulations on the transition matrix of the automaton, the results from [12] follow.

2 Energy Automata

The transition labels on the energy automata which we consider in the paper, will be functions which model transformations of energy levels between system states. Such transformations have the (natural) properties that below a certain energy level, the transition might be disabled (not enough energy is available to perform the transition), and an increase in input energy always yields at least the same increase in output energy. Thus the following definition:

*The work of the first author was supported by the National Foundation of Hungary for Scientific Research, Grant no. K 108448. The work of the second and third authors was supported by ANR MALTHY, grant no. ANR-13-INSE-0003 from the French National Research Foundation, and by the EU FP7 SENSATION project, grant no. 318490 (FP7-ICT-2011-8).

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 48–59, doi:10.4204/EPTCS.191.6

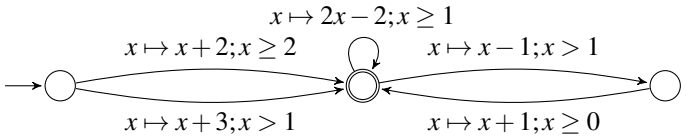


Figure 1: A simple energy automaton.

Definition 1 An *energy function* is a partial function $f : \mathbb{R}_{\geq 0} \rightharpoonup \mathbb{R}_{\geq 0}$ which is defined on a closed interval $[l_f, \infty[$ or on an open interval $]l_f, \infty[$, for some lower bound $l_f \geq 0$, and such that for all $x \leq y$ for which f is defined,

$$yf \geq xf + y - x. \tag{1}$$

The class of all energy functions is denoted by $\mathscr{F}$.

Note that we write function composition and application in diagrammatical order, *from left to right*, in this paper. Hence we write $f;g$, or simply fg, for the composition $g \circ f$ and $x;f$ or xf for function application $f(x)$. This is because we will be concerned with *algebras* of functions, in which function composition is multiplication, and where it is customary to write multiplication in diagrammatical order.

Thus energy functions are strictly increasing, and in points where they are differentiable, the derivative is at least 1. The inverse functions to energy functions exist, but are generally not energy functions. Energy functions can be *composed*, where it is understood that for a composition fg, the interval of definition is $\{x \in \mathbb{R}_{\geq 0} \mid xf \text{ and } xfg \text{ defined}\}$.

Lemma 1 *Let $f \in \mathscr{F}$ and $x \in \mathbb{R}_{\geq 0}$. If $xf < x$, then there is $N \in \mathbb{N}$ for which xf^N is not defined. If $xf > x$, then for all $P \in \mathbb{R}$ there is $N \in \mathbb{N}$ for which $xf^N \geq P$.*

Proof In the first case, we have $x - xf = M > 0$. Using (1), we see that $xf^{n+1} \leq xf^n - M$ for all $n \in \mathbb{N}$ for which xf^{n+1} is defined. Hence $(xf^n)_{n\in\mathbb{N}}$ decreases without bound, so that there must be $N \in \mathbb{N}$ such that xf^N is undefined.

In the second case, we have $xf - x = M > 0$. Again using (1), we see that $xf^{n+1} > xf^n + M$ for all $n \in \mathbb{N}$. Hence $(xf^n)_{n\in\mathbb{N}}$ increases without bound, so that for any $P \in \mathbb{R}$ there must be $N \in \mathbb{N}$ for which $xf^N \geq P$. □

Note that property (1) is not only sufficient for Lemma 1, but in a sense also necessary: if $0 < \alpha < 1$ and $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is the function $xf = 1 + \alpha x$, then $xf^n = \sum_{i=0}^{n-1} \alpha^i + \alpha^n x$ for all $n \in \mathbb{N}$, hence $\lim_{n\to\infty} xf^n = \frac{1}{1-\alpha}$, so Lemma 1 does not hold for f. On the other hand, $yf = xf + \alpha(y-x)$ for all $x \leq y$, so (1) "almost" holds.

Definition 2 An *energy automaton* (S, s_0, T, F) consists of a finite set S of states, with initial state $s_0 \in S$, a finite set $T \subseteq S \times \mathscr{F} \times S$ of transitions labeled with energy functions, and a subset $F \subseteq S$ of acceptance states.

We show an example of a simple energy automaton in Fig. 1. Here we use inequalities to give the definition intervals of energy functions.

A finite *path* in an energy automaton is a finite sequence of transitions $\pi = (s_0, f_1, s_1), (s_1, f_2, s_2), \ldots, (s_{n-1}, f_n, s_n)$. We use f_π to denote the combined energy function $f_1 f_2 \cdots f_n$ of such a finite path. We will also use infinite paths, but note that these generally do not allow for combined energy functions.

A *global state* of an energy automaton is a pair $q = (s, x)$ with $s \in S$ and $x \in \mathbb{R}_{\geq 0}$. A transition between global states is of the form $((s,x), f, (s',x'))$ such that $(s, f, s') \in T$ and $x' = f(x)$. A (finite or infinite) *run* of (S, T) is a path in the graph of global states and transitions.

We are ready to state the decision problems with which our main concern will lie. As the input to a decision problem must be in some way finitely representable, we will state them for subclasses $\mathscr{F}' \subseteq \mathscr{F}$ of *computable* energy functions; an $\mathscr{F}'$-automaton is an energy automaton (S,T) with $T \subseteq S \times \mathscr{F}' \times S$.

Problem 1 (Reachability) Given a subset $\mathscr{F}' \subseteq \mathscr{F}$ of computable functions, an $\mathscr{F}'$-automaton $A = (S,s_0,T,F)$ and a computable initial energy $x_0 \in \mathbb{R}_{\geq 0}$: does there exist a finite run of A from (s_0,x_0) which ends in a state in F?

Problem 2 (Büchi acceptance) Given a subset $\mathscr{F}' \subseteq \mathscr{F}$ of computable functions, an $\mathscr{F}'$-automaton $A = (S,s_0,T,F)$ and a computable initial energy $x_0 \in \mathbb{R}_{\geq 0}$: does there exist an infinite run of A from (s_0,x_0) which visits F infinitely often?

As customary, a run such as in the statements above is said to be accepting.

3 Algebraic Preliminaries

We now turn our attention to the algebraic setting of *-continuous Kleene algebras and related structures, before revisiting energy automata in Section 6. In this section we review some results on *-continuous Kleene algebras and *-continuous Kleene ω-algebras.

3.1 *-Continuous Kleene ω-Algebras

A *semiring* [1, 18] $S = (S,+,\cdot,0,1)$ consists of a commutative monoid $(S,+,0)$ and a monoid $(S,\cdot,1)$ such that the distributive laws

$$x(y+z) = xy+xz$$
$$(y+z)x = yx+zx$$

and the zero laws

$$0 \cdot x = 0 = x \cdot 0$$

hold for all $x,y,z \in S$. It follows that the product operation distributes over all finite sums.

An *idempotent semiring* is a semiring S whose sum operation is idempotent, so that $x+x = x$ for all $x \in S$. Each idempotent semiring S is partially ordered by the relation $x \leq y$ iff $x+y = y$, and then sum and product preserve the partial order and 0 is the least element. Moreover, for all $x,y \in S$, $x+y$ is the least upper bound of the set $\{x,y\}$. Accordingly, in an idempotent semiring S, we will usually denote the sum operation by $\vee$ and 0 by $\bot$.

A *Kleene algebra* [22] is an idempotent semiring $S = (S,\vee,\cdot,\bot,1)$ equipped with a star operation $^* : S \to S$ such that for all $x,y \in S$, yx^* is the least solution of the fixed point equation $z = zx \vee y$ and x^*y is the least solution of the fixed point equation $z = xz \vee y$ with respect to the natural order.

A **-continuous Kleene algebra* [22] is a Kleene algebra $S = (S,\vee,\cdot,^*,\bot,1)$ in which the infinite suprema $\bigvee\{x^n \mid n \geq 0\}$ exist for all $x \in S$, $x^* = \bigvee\{x^n \mid n \geq 0\}$ for every $x \in S$, and product preserves such suprema:

$$y(\bigvee_{n\geq 0} x^n) = \bigvee_{n\geq 0} yx^n \quad \text{and} \quad (\bigvee_{n\geq 0} x^n)y = \bigvee_{n\geq 0} x^n y$$

for all $x,y \in S$.

A *continuous Kleene algebra* is a Kleene algebra $S = (S,\vee,\cdot,^*,\bot,1)$ in which *all* suprema $\bigvee X$, $X \subseteq S$, exist and are preserved by products, *i.e.,* $y(\bigvee X) = \bigvee yX$ and $(\bigvee X)y = \bigvee Xy$ for all $X \subseteq S$, $y \in$

S. *-continuous Kleene algebras are hence a generalization of continuous Kleene algebras. There are interesting Kleene algebras which are *-continuous but not continuous, for example the Kleene algebra of all regular languages over some alphabet.

A *semiring-semimodule pair* [2,14] (S,V) consists of a semiring $S=(S,+,\cdot,0,1)$ and a commutative monoid $V=(V,+,0)$ which is equipped with a left S-action $S\times V\to V$, $(s,v)\mapsto sv$, satisfying

$$\begin{array}{cc} (s+s')v=sv+s'v & s(v+v')=sv+sv' \\ (ss')v=s(s'v) & 0s=0 \\ s0=0 & 1v=v \end{array}$$

for all $s,s'\in S$ and $v\in V$. In that case, we also call V a *(left) S-semimodule*. If S is idempotent, then also V is idempotent, so that we then write $V=(V,\vee,\bot)$.

A *generalized *-continuous Kleene algebra* [10] is a semiring-semimodule pair (S,V) where $S=(S,\vee,\cdot,^*,\bot,1)$ is a *-continuous Kleene algebra such that

$$xy^*v=\bigvee_{n\geq 0} xy^nv$$

for all $x,y\in S$ and $v\in V$.

A **-continuous Kleene ω-algebra* [10] consists of a generalized *-continuous Kleene algebra (S,V) together with an infinite product operation $S^\omega\to V$ which maps every infinite sequence $x_0,x_1,\dots$ in S to an element $\prod_{n\geq 0}x_n$ of V. The infinite product is subject to the following conditions:

(C1) For all $x_0,x_1,\dots\in S$, $\displaystyle\prod_{n\geq 0}x_n=x_0\prod_{n\geq 0}x_{n+1}$.

(C2) Let $x_0,x_1,\dots\in S$ and $0=n_0\leq n_1\leq\cdots$ a sequence which increases without a bound. Let $y_k=x_{n_k}\cdots x_{n_{k+1}-1}$ for all $k\geq 0$. Then $\displaystyle\prod_{n\geq 0}x_n=\prod_{k\geq 0}y_k$.

(C3) For all $x_0,x_1,\dots,y,z\in S$, $\displaystyle\prod_{n\geq 0}(x_n(y\vee z))=\bigvee_{x'_0,x'_1,\dots\in\{y,z\}}\prod_{n\geq 0}x_nx'_n$.

(C4) For all $x,y_0,y_1,\dots\in S$, $\displaystyle\prod_{n\geq 0}x^*y_n=\bigvee_{k_0,k_1,\dots\geq 0}\prod_{n\geq 0}x^{k_n}y_n$.

A *continuous Kleene ω-algebra* [14] is a semiring-semimodule pair (S,V) in which S is a continuous Kleene algebra, V is a complete lattice, and the S-action on V preserves all suprema in either argument, together with an infinite product as above which satisfies conditions (C1) and (C2) above and preserves all suprema: $\prod_{n\geq 0}(\bigvee X_n)=\bigvee\{\prod_{n\geq 0}x_n\mid x_n\in X_n,n\geq 0\}$ for all $X_0,X_1,\dots\subseteq S$ (this property implies (C3) and (C4) above). *-continuous Kleene ω-algebras are hence a generalization of continuous Kleene ω-algebras. We have in [10] given an example, based on regular languages of finite and infinite words, of a *-continuous Kleene ω-algebra which is not a continuous Kleene ω-algebra. In Section 6 we will show that energy functions give raise to another such example.

3.2 Matrix Semiring-Semimodule Pairs

For any semiring S and $n\geq 1$, we can form the matrix semiring $S^{n\times n}$ whose elements are $n\times n$-matrices of elements of S and whose sum and product are given as the usual matrix sum and product. It is known [21]

that when S is a *-continuous Kleene algebra, then $S^{n\times n}$ is also a *-continuous Kleene algebra, with the *-operation defined by

$$M^*_{i,j} = \bigvee_{m\geq 0} \bigvee_{1\leq k_1,\ldots,k_m\leq n} M_{i,k_1}M_{k_1,k_2}\cdots M_{k_m,j}$$

for all $M \in S^{n\times n}$ and $1 \leq i,j \leq n$. The above infinite supremum exists, as it is taken over a regular set, see [13, Thm. 9] and [10, Lemma 4]. Also, if $n \geq 2$ and $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, where a and d are square matrices of dimension less than n, then

$$M^* = \begin{pmatrix} (a \vee bd^*c)^* & (a \vee bd^*c)^*bd^* \\ (d \vee ca^*b)^*ca^* & (d \vee ca^*b)^* \end{pmatrix}. \tag{2}$$

For any semiring-semimodule pair (S,V) and $n \geq 1$, we can form the matrix semiring-semimodule pair $(S^{n\times n}, V^n)$ whose elements are $n \times n$-matrices of elements of S and n-dimensional (column) vectors of elements of V, with the action of $S^{n\times n}$ on V^n given by the usual matrix-vector product.

When (S,V) is a *-continuous Kleene ω-algebra, then $(S^{n\times n}, V^n)$ is a generalized *-continuous Kleene algebra [10]. By [10, Lemma 17], there is an ω-operation on $S^{n\times n}$ defined by

$$M^\omega_i = \bigvee_{1\leq k_1,k_2,\ldots\leq n} M_{i,k_1}M_{k_1,k_2}\cdots$$

for all $M \in S^{n\times n}$ and $1 \leq i \leq n$. Also, if $n \geq 2$ and $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, where a and d are square matrices of dimension less than n, then

$$M^\omega = \begin{pmatrix} (a \vee bd^*c)^\omega \vee (a \vee bd^*c)^*bd^\omega \\ (d \vee ca^*b)^\omega \vee (d \vee ca^*b)^*ca^\omega \end{pmatrix}.$$

3.3 Weighted automata

Let (S,V) be a *-continuous Kleene ω-algebra and $A \subseteq S$ a subset. We write $\langle A\rangle$ for the set of all finite suprema $a_1 \vee \cdots \vee a_m$ with $a_i \in A$ for each $i = 1,\ldots,m$.

A *weighted automaton* [15] over A of dimension $n \geq 1$ is a tuple (α, M, k), where $\alpha \in \{\bot,1\}^n$ is the initial vector, $M \in \langle A\rangle^{n\times n}$ is the transition matrix, and k is an integer $0 \leq k \leq n$. Combinatorially, this may be represented as a transition system whose set of states is $\{1,\ldots,n\}$. For any pair of states i,j, the transitions from i to j are determined by the entry $M_{i,j}$ of the transition matrix: if $M_{i,j} = a_1 \vee \cdots \vee a_m$, then there are m transitions from i to j, respectively labeled $a_1,\ldots,a_n$. The states i with $\alpha_i = 1$ are *initial*, and the states $\{1,\ldots,k\}$ are *accepting*.

The *finite behavior* of a weighted automaton $A = (\alpha, M, k)$ is defined to be

$$|A| = \alpha M^* \kappa,$$

where $\kappa \in \{\bot,1\}^n$ is the vector given by $\kappa_i = 1$ for $i \leq k$ and $\kappa_i = \bot$ for $i > k$. (Note that α has to be used as a *row* vector for this multiplication to make sense.) It is clear by (2) that $|A|$ is the supremum of the products of the transition labels along all paths in A from any initial to any accepting state.

The *Büchi behavior* of a weighted automaton $A = (\alpha, M, k)$ is defined to be

$$\|A\| = \alpha \begin{pmatrix} (a+bd^*c)^\omega \\ d^*c(a+bd^*c)^\omega \end{pmatrix},$$

where $a \in \langle A\rangle^{k\times k}$, $b \in \langle A\rangle^{k\times(n-k)}$, $c \in \langle A\rangle^{(n-k)\times n}$ and $d \in \langle A\rangle^{(n-k)\times(n-k)}$ are such that $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$. By [10, Thm. 20], $\|A\|$ is the supremum of the products of the transition labels along all infinite paths in A from any initial state which infinitely often visit an accepting state.

4 Generalized *-continuous Kleene Algebras of Functions

In the following two sections our aim is to establish properties which ensure that semiring-semimodule pairs of functions form *-continuous Kleene ω-algebras. We will use these properties in Section 6 to show that energy functions form a *-continuous Kleene ω-algebra.

Let L and L' be complete lattices with bottom and top elements $\bot$ and $\top$. Then a function $f : L \to L'$ is said to be *finitely additive* if $\bot f = \bot$ and $(x \vee y)f = xf \vee yf$ for all $x, y \in L$. (Recall that we write function application and composition in the diagrammatic order, from left to right.) When $f : L \to L'$ is finitely additive, then $(\bigvee X)f = \bigvee Xf$ for all finite sets $X \subseteq L$.

Consider the collection $\mathrm{FinAdd}_{L,L'}$ of all finitely additive functions $f : L \to L'$, ordered pointwise. Since the (pointwise) supremum of any set of finitely additive functions is finitely additive, $\mathrm{FinAdd}_{L,L'}$ is also a complete lattice, in which the supremum of any set of functions can be constructed pointwise. The least and greatest elements are the constant functions with value $\bot$ and $\top$, respectively. By an abuse of notation, we will denote these functions by $\bot$ and $\top$ as well.

Definition 3 A function $f \in \mathrm{FinAdd}_{L,L'}$ is said to be $\top$-*continuous* if $f = \bot$ or for all $X \subseteq L$ with $\bigvee X = \top$, also $\bigvee Xf = \top$.

Note that if $f \neq \bot$ is $\top$-continuous, then $\top f = \top$. The functions id and $\bot$ are $\top$-continuous. Also, the (pointwise) supremum of any set of $\top$-continuous functions is again $\top$-continuous.

We will first be concerned with functions in $\mathrm{FinAdd}_{L,L}$, which we just denote FinAdd_L. Since the composition of finitely additive functions is finitely additive and the identity function id over L is finitely additive, and since composition of finitely additive functions distributes over finite suprema, FinAdd_L, equipped with the operation $\vee$ (binary supremum), ; (composition), and the constant function $\bot$ and the identity function id as 1, is an idempotent semiring. It follows that when f is finitely additive, then so is $f^* = \bigvee_{n\geq 0} f^n$. Moreover, $f \leq f^*$ and $f^* \leq g^*$ whenever $f \leq g$. Below we will usually write just fg for the composition $f;g$.

Lemma 2 *Let S be any subsemiring of* FinAdd_L *closed under the *-operation. Then S is a *-continuous Kleene algebra iff for all $g, h \in S$, $g^*h = \bigvee_{n\geq 0} g^n h$.*

Proof Suppose that the above condition holds. We need to show that $f(\bigvee_{n\geq 0} g^n)h = \bigvee_{n\geq 0} fg^n h$ for all $f, g, h \in S$. But $f(\bigvee_{n\geq 0} g^n)h = f(\bigvee_{n\geq 0} g^n h)$ by assumption, and we conclude that $f(\bigvee_{n\geq 0} g^n h) = \bigvee_{n\geq 0} fg^n h$ since the supremum is pointwise. □

Compositions of $\top$-continuous functions in FinAdd_L are again $\top$-continuous, so that the collection of all $\top$-continuous functions in FinAdd_L is itself an idempotent semiring.

Definition 4 A function $f \in \mathrm{FinAdd}_L$ is said to be *locally *-closed* if for each $x \in L$, either $xf^* = \top$ or there exists $N \geq 0$ such that $xf^* = x \vee \cdots \vee xf^N$.

The functions id and $\bot$ are locally *-closed. As the next example demonstrates, compositions of locally *-closed (and $\top$-continuous) functions are not necessarily locally *-closed.

Example 1 Let L be the following complete lattice (the linear sum of three infinite chains):

$$\bot < x_0 < x_1 < \cdots < y_0 < y_1 < \cdots < z_0 < z_1 < \cdots < \top$$

Since L is a chain, a function $L \to L$ is finitely additive iff it is monotone and preserves $\bot$.

Let $f, g : L \to L$ be the following functions. First, $\bot f = \bot g = \bot$ and $\top f = \top g = \top$. Moreover, $x_i f = y_i$, $y_i f = z_i g = \top$ and $x_i g = \bot$, $y_i g = x_{i+1}$, and $z_i g = \top$ for all i. Then f, g are monotone, $uf^* =$

$u \vee uf \vee uf^2$ and $ug^* = u \vee ug$ for all $u \in L$. Also, f and g are $\top$-continuous, since if $\bigvee X = \top$ then either $\top \in X$ or $X \cap \{z_0, z_1, \dots\}$ is infinite, but then $\bigvee Xf = \bigvee Xg = \top$. However, fg is not locally *-closed, since $x_0(fg)^* = x_0 \vee x_0(fg) \vee x_0(fg)^2 \cdots = x_0 \vee x_1 \vee \cdots = y_0$. □

Lemma 3 *Let $f \in \mathrm{FinAdd}_L$ be locally *-closed. Then also f^* is locally *-closed. If f is additionally $\top$-continuous, then so is f^*.*

Proof We prove that $xf^{**} = x \vee xf^* = xf^*$ for all $x \in L$. Indeed, this is clear when $xf^* = \top$, since $f^* \leq f^{**}$. Otherwise $xf^* = \bigvee_{k \leq n} xf^k$ for some $n \geq 0$.

By finite additivity, it follows that $xf^*f^* = \bigvee_{k \leq n} xf^k f^*$. But for each k, $xf^k f^* = xf^k \vee xf^{k+1} \vee \cdots \leq xf^*$, thus $xf^* = xf^*f^*$ and $xf^* = xf^{**}$. It follows that f^* is locally *-closed.

Suppose now that f is additionally $\top$-continuous. We need to show that f^* is also $\top$-continuous. To this end, let $X \subseteq L$ with $\bigvee X = \top$. Since $x \leq xf^*$ for all $x \in X$, it holds that $\bigvee Xf^* \geq \bigvee X = \top$. Thus $\bigvee Xf^* = \top$. □

Proposition 4 *Let S be any subsemiring of FinAdd_L closed under the *-operation. If each $f \in S$ is locally *-closed and $\top$-continuous, then S is a *-continuous Kleene algebra.*

Proof Suppose that $g, h \in S$. By Lemma 2, it suffices to show that $g^*h = \bigvee_{n \geq 0} g^n h$. Since this is clear when $h = \bot$, assume that $h \neq \bot$. As $g^n h \leq g^* h$ for all $n \geq 0$, it holds that $\bigvee_{n \geq 0} g^n h \leq g^* h$. To prove the opposite inequality, suppose that $x \in L$. If $xg^* = \top$, then $\bigvee_{n \geq 0} xg^n = \top$, so $\bigvee_{n \geq 0} xg^n h = \top$ by $\top$-continuity. Thus, $xg^*h = \top = \bigvee_{n \geq 0} xg^n h$.

Suppose that $xg^* \neq \top$. Then there is $m \geq 0$ with

$$xg^*h = (x \vee \cdots \vee xg^m)h = xh \vee \cdots \vee xg^m h \leq \bigvee_{n \geq 0} xg^n h = x(\bigvee_{n \geq 0} g^n h)\,. \qquad \square$$

Now define a left action of FinAdd_L on $\mathrm{FinAdd}_{L,L'}$ by $fv = f;v$, for all $f \in \mathrm{FinAdd}_L$ and $v \in \mathrm{FinAdd}_{L,L'}$. It is a routine matter to check that $\mathrm{FinAdd}_{L,L'}$, equipped with the above action, the binary supremum operation $\vee$ and the constant $\bot$ is an (idempotent) left FinAdd_L-semimodule, that is, $(\mathrm{FinAdd}_L, \mathrm{FinAdd}_{L,L'})$ is a semiring-semimodule pair.

Lemma 5 *Let $S \subseteq \mathrm{FinAdd}_L$ be a *-continuous Kleene algebra and $V \subseteq \mathrm{FinAdd}_{L,L'}$ an S-semimodule. Then (S,V) is a generalized *-continuous Kleene algebra iff for all $f \in S$ and $v \in V$, $f^*v = \bigvee_{n \geq 0} f^n v$.*

Proof Similar to the proof of Lemma 2 □

Proposition 6 *Let $S \subseteq \mathrm{FinAdd}_L$ be a *-continuous Kleene algebra and $V \subseteq \mathrm{FinAdd}_{L,L'}$ an S-semimodule. If each $f \in S$ is locally *-closed and $\top$-continuous and each $v \in V$ is $\top$-continuous, then (S,V) is a generalized *-continuous Kleene algebra.*

Proof Similar to the proof of Proposition 4. □

5 *-continuous Kleene ω-Algebras of Functions

In this section, let L be an arbitrary complete lattice and $L' = \mathbf{2}$, the 2-element lattice $\{\bot, \top\}$. We define an infinite product $\mathrm{FinAdd}_L^\omega \to \mathrm{FinAdd}_{L,\mathbf{2}}$. Let $f_0, f_1, \dots \in \mathrm{FinAdd}_L$ be an infinite sequence and define $v = \prod_{n \geq 0} f_n : L \to \mathbf{2}$ by

$$xv = \begin{cases} \bot & \text{if there is } n \geq 0 \text{ such that } xf_0 \cdots f_n = \bot, \\ \top & \text{otherwise} \end{cases}$$

for all $x \in L$. We will write $\prod_{n\geq k} f_n$, for $k \geq 0$, as a shorthand for $\prod_{n\geq 0} f_{n+k}$.

It is easy to see that $\prod_{n\geq 0} f_n$ is finitely additive. Indeed, $\bot \prod_{n\geq 0} f_n = \bot$ clearly holds, and for all $x \leq y \in L$, $x\prod_{n\geq 0} f_n \leq y\prod_{n\geq 0} f_n$. Thus, to prove that $(x\vee y)\prod_{n\geq 0} f_n = x\prod_{n\geq 0} f_n \vee y\prod_{n\geq 0} f_n$ for all $x,y \in L$, it suffices to show that if $x\prod_{n\geq 0} f_n = y\prod_{n\geq 0} f_n = \bot$, then $(x\vee y)\prod_{n\geq 0} f_n = \bot$. But if $x\prod_{n\geq 0} f_n = y\prod_{n\geq 0} f_n = \bot$, then there exist $m,k \geq 0$ such that $xf_0\cdots f_m = yf_0\cdots f_k = \bot$. Let $n = \max\{m,k\}$. We have $(x\vee y)f_0\cdots f_n = xf_0\cdots f_n \vee yf_0\cdots f_n = \bot$, and thus $(x\vee y)\prod_{n\geq 0} f_n = \bot$.

It is clear that this infinite product satisfies conditions (C1) and (C2) in the definition of *-continuous Kleene ω-algebra. Below we show that also (C3) and (C4) hold.

Lemma 7 *For all $f_0, f_1, \ldots, g_0, g_1, \ldots \in \mathrm{FinAdd}_L$,*

$$\prod_{n\geq 0}(f_n \vee g_n) = \bigvee_{h_n\in\{f_n,g_n\}} \prod_{n\geq 0} h_n\,.$$

Proof Since infinite product is monotone, the term on the right-hand side of the equation is less than or equal to the term on the left-hand side. To prove that equality holds, let $x \in L$ and suppose that $x\prod_{n\geq 0}(f_n \vee g_n) = \top$. It suffices to show that there is a choice of the functions $h_n \in \{f_n, g_n\}$ such that $x\prod_{n\geq 0} h_n = \top$.

Consider the infinite ordered binary tree where each node at level $n \geq 0$ is the source of an edge labeled f_n and an edge labeled g_n, ordered as indicated. We can assign to each node u the composition h_u of the functions that occur as the labels of the edges along the unique path from the root to that node.

Let us mark a node u if $xh_u \neq \bot$. As $x\prod_{n\geq 0}(f_n \vee g_n) = \top$, each level contains a marked node. Moreover, whenever a node is marked and has a predecessor, its predecessor is also marked. By König's lemma [20] there is an infinite path going through marked nodes. This infinite path gives rise to the sequence $h_0, h_1, \ldots$ with $x\prod_{n\geq 0} h_n = \top$. □

Lemma 8 *Let $f \in \mathrm{FinAdd}_L$ and $v \in \mathrm{FinAdd}_{L,\mathbf{2}}$ such that f is locally *-closed and v is $\top$-continuous. If $xf^*v = \top$, then there exists $k \geq 0$ such that $xf^kv = \top$.*

Proof If $xf^* = \bigvee_{n=0}^N xf^n$ for some $N \geq 0$, then $xf^*v = \bigvee_{n=0}^N xf^nv = \top$ implies the claim of the lemma. If $xf^* = \top$, then $\top$-continuity of v implies that $\bigvee_{n\geq 0} xf^nv = \top$, which again implies the claim. □

Lemma 9 *Let $f, g_0, g_1, \ldots \in \mathrm{FinAdd}_L$ be locally *-closed and $\top$-continuous such that for each $m \geq 0$, $g_m\prod_{n\geq m+1} f^*g_n \in \mathrm{FinAdd}_{L,\mathbf{2}}$ is $\top$-continuous. Then*

$$\prod_{n\geq 0} f^*g_n = \bigvee_{k_0,k_1,\ldots\geq 0} \prod_{n\geq 0} f^{k_n}g_n\,.$$

Proof As infinite product is monotone, the term on the right-hand side of the equation is less than or equal to the term on the left-hand side. To prove that equality holds, let $x \in L$ and suppose that $x\prod_{n\geq 0} f^*g_n = \top$. We want to show that there exist integers $k_0, k_1, \ldots \geq 0$ such that $x\prod_{n\geq 0} f^{k_n}g_n = \top$.

Let $x_0 = x$. By Lemma 8, $x\prod_{n\geq 0} f^*g_n = x_0f^*g_0\prod_{n\geq 1} f^*g_n = \top$ implies that there is $k_0 \geq 0$ for which $x_0f^{k_0}g_0\prod_{n\geq 1} f^*g_n = \top$. We finish the proof by induction. Assume we have $k_0, \ldots, k_m \geq 0$ such that $xf^{k_0}g_0\cdots f^{k_m}g_m\prod_{n\geq m+1} f^*g_n = \top$ and let $x_{m+1} = xf^{k_0}g_0\cdots f^{k_m}g_m$. Then $x_{m+1}f^*g_{m+1}\prod_{n\geq m+2} f^*g_n = \top$ implies, using Lemma 8, that there exists $k_{m+1} \geq 0$ for which $x_{m+1}f^{k_{m+1}}g_{m+1}\prod_{n\geq m+2} f^*g_n = \top$. □

Proposition 10 *Let $S \subseteq \mathrm{FinAdd}_L$ and $V \subseteq \mathrm{FinAdd}_{L,\mathbf{2}}$ such that (S,V) is a generalized *-continuous Kleene algebra of locally *-closed and $\top$-continuous functions $L \to L$ and $\top$-continuous functions $L \to \mathbf{2}$. If $\prod_{n\geq 0} f_n \in V$ for all sequences $f_0, f_1, \ldots$ of functions in S, then (S,V) is a *-continuous Kleene ω-algebra.*

Proof This is clear from Lemmas 7 and 9. □

We finish the section by a lemma which exhibits a condition on the lattice L which ensures that infinite products of locally *-closed and $\top$-continuous functions are again $\top$-continuous.

Lemma 11 *Assume that L has the property that whenever $\bigvee X = \top$ for some $X \subseteq L$, then for all $x < \top$ in L there is $y \in X$ with $x \leq y$. If $f_0, f_1, \ldots \in \mathrm{FinAdd}_L$ is a sequence of locally *-closed and $\top$-continuous functions, then $\prod_{n\geq 0} f_n \in \mathrm{FinAdd}_{L,\mathbf{2}}$ is $\top$-continuous.*

Proof Let $v = \prod_{n\geq 0} f_n$. We already know that v is finitely additive. We need to show that if $v \neq \bot$, then v is $\top$-continuous. But if $v \neq \bot$, then there is some $x < \top$ with $xv = \top$, *i.e.,* such that $xf_0 \cdots f_n > \bot$ for all n. By assumption, there is some $y \in X$ with $x \leq y$. It follows that $yf_0 \cdots f_n \geq xf_0 \cdots f_n > \bot$ for all n and thus $\bigvee Xv = \top$. □

6 Energy Automata Revisited

We finish this paper by showing how the setting developed in the last sections can be applied to solve the energy problems of Section 2. Let $L = [0, \top]_\bot$ be the complete lattice of nonnegative real numbers together with $\top = \infty$ and an extra bottom element $\bot$, and extend the usual order and operations on real numbers to L by declaring that $\bot < x < \top$, $\bot - x = \bot$ and $\top + x = \top$ for all $x \in \mathbb{R}_{\geq 0}$. Note that L satisfies the precondition of Lemma 11.

We extend the definition of energy function:

Definition 5 An *extended energy function* is a mapping $f : L \to L$ for which $\bot f = \bot$, $\top f = \bot$ if $xf = \bot$ for all $x < \top$ and $\top f = \top$ otherwise, and $yf \geq xf + y - x$ whenever $\bot < x < y < \top$. The set of such functions is denoted $\mathscr{E}$.

Every energy function $f : \mathbb{R}_{\geq 0} \rightharpoonup \mathbb{R}_{\geq 0}$ as of Definition 1 gives rise to an extended energy function $\tilde{f} : L \to L$ given by $\bot\tilde{f} = \bot$, $x\tilde{f} = \bot$ if xf is undefined, $x\tilde{f} = xf$ otherwise for $x \in \mathbb{R}_{\geq 0}$, and $\top\tilde{f} = \top$. This defines an embedding $\mathscr{F} \hookrightarrow \mathscr{E}$.

The definition entails that for all $f \in \mathscr{E}$ and all $x < y \in L$, $xf = \top$ implies $yf = \top$ and $yf = \bot$ implies $xf = \bot$. Note that $\mathscr{E}$ is closed under (pointwise) binary supremum $\vee$ and composition and contains the functions $\bot$ and id.

Lemma 12 *Extended energy functions are finitely additive and $\top$-continuous, hence $\mathscr{E} \subseteq \mathrm{FinAdd}_L$ is a semiring.*

Proof Finite additivity follows from monotonicity. For $\top$-continuity, let $X \subseteq L$ such that $\bigvee X = \top$ and $f \in \mathscr{E}$, $f \neq \bot$. We have $X \neq \{\bot\}$, so let $x_0 \in X \setminus \{\bot\}$ and, for all $n \geq 0$, $x_n = x_0 + n$. Let $y_n = x_n f$. If $y_n = \bot$ for all $n \geq 0$, then also $nf = \bot$ for all $n \geq 0$ (as $x_n \geq n$), hence $f = \bot$. We must thus have an index N for which $y_N > \bot$. But then $y_{N+k} \geq y_N + k$ for all $k \geq 0$, hence $\bigvee Xf = \top$. □

Lemma 13 *For $f \in \mathscr{E}$, f^* is given by $xf^* = x$ if $xf \leq x$ and $xf^* = \top$ if $xf > x$. Hence f is locally *-closed and $f^* \in \mathscr{E}$.*

Proof We have $\bot f^* = \bot$ and $\top f^* = \top$. Let $x \neq \bot, \top$. If $xf \leq x$, then $xf^n \leq x$ for all $n \geq 0$, so that $x \leq \bigvee_{n\geq 0} xf^n \leq x$, whence $xf^* = x$. If $xf > x$, then let $a = xf - x > 0$. We have $xf \geq x + a$, hence $xf^n \geq x + na$ for all $n \geq 0$, so that $xf^* = \bigvee_{n\geq 0} xf^n = \top$. □

Not all locally *-closed functions $f : L \to L$ are energy functions: the function f defined by $xf = 1$ for $x < 1$ and $xf = x$ for $x \geq 1$ is locally *-closed, but $f \notin \mathscr{E}$.

Corollary 14 *$\mathscr{E}$ is a *-continuous Kleene algebra.*

Proof This is clear by Proposition 4. □

Remark It is *not* true that $\mathscr{E}$ is a *continuous* Kleene algebra: Let $f_n, g \in \mathscr{E}$ be defined by $xf_n = x+1-\frac{1}{n+1}$ for $x \geq 0$, $n \geq 0$ and $xg = x$ for $x \geq 1$, $xg = \bot$ for $x < 1$. Then $0(\bigvee_{n\geq 0} f_n)g = (\bigvee_{n\geq 0} 0f_n)g = 1g = 1$, whereas $0\bigvee_{n\geq 0}(f_n g) = \bigvee_{n\geq 0}(0f_n g) = \bigvee_{n\geq 0}((1-\frac{1}{n+1})g) = \bot$.

Let $\mathscr{V}$ denote the $\mathscr{E}$-semimodule of all $\top$-continuous functions $L \to \mathbf{2}$. For $f_0, f_1, \ldots \in \mathscr{E}$, define the infinite product $f = \prod_{n\geq 0} f_n : L \to \mathbf{2}$ by $xf = \bot$ if there is an index n for which $xf_0 \cdots f_n = \bot$ and $xf = \top$ otherwise, like in Section 5. By Lemma 11, $\prod_{n\geq 0} f_n$ is $\top$-continuous, *i.e.,* $\prod_{n\geq 0} f_n \in \mathscr{V}$.

By Proposition 6, $(\mathscr{E}, \mathscr{V})$ is a generalized *-continuous Kleene algebra.

Corollary 15 *$(\mathscr{E}, \mathscr{V})$ is a *-continuous Kleene ω-algebra.*

Proof This is clear by Proposition 10. □

Remark As $\mathscr{E}$ is not a continuous Kleene algebra, it also holds that $(\mathscr{E}, \mathscr{V})$ is not a continuous Kleene ω-algebra; in fact it is clear that there is no $\mathscr{E}$-semimodule $\mathscr{V}'$ for which $(\mathscr{E}, \mathscr{V})$ would be a continuous Kleene ω-algebra. The initial motivation for the work in [10] and the present paper was to generalize the theory of continuous Kleene ω-algebras so that it would be applicable to energy functions.

Noting that energy automata are weighted automata over $\mathscr{E}$ in the sense of Section 3.3, we can now solve the reachability and Büchi problem for energy automata:

Theorem 1 *Let $A = (\alpha, M, k)$ be an energy automaton and $x_0 \in \mathbb{R}_{\geq 0}$. There exists a finite run of A from an initial state to an accepting state with initial energy x_0 iff $x_0|A| > \bot$.*

Theorem 2 *Let $A = (\alpha, M, k)$ be an energy automaton and $x_0 \in \mathbb{R}_{\geq 0}$. There exists an infinite run of A from an initial state which infinitely often visits an accepting state iff $x_0\|A\| = \top$.*

Corollary 16 *Problems 1 and 2 are decidable.*

In [12], the complexity of the decision procedure has been established for important subclasses of energy functions.

7 Conclusion and Further Work

We have shown that energy functions form a *-continuous Kleene ω-algebra [10], hence that *-continuous Kleene ω-algebras provide a proper algebraic setting for energy problems. On our way, we have proven more general results about properties of finitely additive functions on complete lattices which should be of a more general interest.

There are interesting generalizations of our setting of energy automata which, we believe, can be attacked using techniques similar to ours. One such generalization are energy problems for *real time* or *hybrid* models, as for example treated in [3–5, 23]. Another generalization is to higher dimensions, like in [16, 19, 24] and other papers.

References

[1] Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series With Applications*. Cambridge Univ. Press, 2010.

[2] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories: The Equational Logic of Iterative Processes*. EATCS monographs on theoretical computer science. Springer-Verlag, 1993. http://dx.doi.org/10.1007/978-3-642-78034-9.

[3] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 61–70. ACM, 2010. http://doi.acm.org/10.1145/1755952.1755963.

[4] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lect. Notes Comput. Sci.*, pages 33–47. Springer-Verlag, 2008. http://dx.doi.org/10.1007/978-3-540-85778-5_4.

[5] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Lower-bound-constrained runs in weighted timed automata. *Perform. Eval.*, 73:91–109, 2014. http://dx.doi.org/10.1016/j.peva.2013.11.002.

[6] Romain Brenguier, Franck Cassez, and Jean-François Raskin. Energy and mean-payoff timed games. In Martin Fränzle and John Lygeros, editors, *HSCC*, pages 283–292. ACM, 2014. http://doi.acm.org/10.1145/2562059.2562116.

[7] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. http://dx.doi.org/10.1016/j.tcs.2012.07.038.

[8] Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lect. Notes Comput. Sci.*, pages 260–274. Springer-Verlag, 2010. http://dx.doi.org/10.1007/978-3-642-15205-4_22.

[9] Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 2009.

[10] Zoltán Ésik, Uli Fahrenberg, and Axel Legay. *-continuous Kleene ω-algebras. In Igor Potapov, editor, *DLT*, volume 9168 of *Lect. Notes Comput. Sci.*, pages 240–251. Springer-Verlag, 2015. http://dx.doi.org/10.1007/978-3-319-21500-6_19.

[11] Zoltán Ésik, Uli Fahrenberg, and Axel Legay. *-continuous Kleene ω-algebras. *CoRR*, abs/1501.01118, 2015. http://arxiv.org/abs/1501.01118.

[12] Zoltán Ésik, Uli Fahrenberg, Axel Legay, and Karin Quaas. Kleene algebras and semimodules for energy problems. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA*, volume 8172 of *Lect. Notes Comput. Sci.*, pages 102–117. Springer-Verlag, 2013. http://dx.doi.org/10.1007/978-3-319-02444-8_9.

[13] Zoltán Ésik and Werner Kuich. Rationally additive semirings. *J. Univ. Comput. Sci.*, 8(2):173–183, 2002. http://dx.doi.org/10.3217/jucs-008-02-0173.

[14] Zoltán Ésik and Werner Kuich. On iteration semiring-semimodule pairs. *Semigroup Forum*, 75:129–159, 2007.

[15] Zoltán Ésik and Werner Kuich. *Finite Automata*, chapter 3. In Droste et al. [9], 2009.

[16] Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jiří Srba. Energy games in multiweighted automata. In *ICTAC*, volume 6916 of *Lect. Notes Comput. Sci.*, pages 95–115. Springer-Verlag, 2011. http://dx.doi.org/10.1007/978-3-642-23283-1_9.

[17] Uli Fahrenberg, Axel Legay, and Karin Quaas. Büchi conditions for generalized energy automata. In *WATA*, page 47, 2012.

[18] Jonathan S. Golan. *Semirings and their Applications*. Springer-Verlag, 1999.

[19] Line Juhl, Kim G. Larsen, and Jean-François Raskin. Optimal bounds for multiweighted and parametrised energy games. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods*, volume 8051 of *Lect. Notes Comput. Sci.*, pages 244–255. Springer-Verlag, 2013. http://dx.doi.org/10.1007/978-3-642-39698-4_15.

[20] Dénes König. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3(2-3):121–130, 1927.

[21] Dexter Kozen. On Kleene algebras and closed semirings. In Branislav Rovan, editor, *MFCS*, volume 452 of *Lect. Notes Comput. Sci.*, pages 26–47. Springer-Verlag, 1990. http://dx.doi.org/10.1007/BFb0029594.

[22] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. http://dx.doi.org/10.1006/inco.1994.1037.

[23] Karin Quaas. On the interval-bound problem for weighted timed automata. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *Lect. Notes Comput. Sci.*, pages 452–464. Springer-Verlag, 2011. http://dx.doi.org/10.1007/978-3-642-21254-3_36.

[24] Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. http://dx.doi.org/10.1016/j.ic.2015.03.001.

Self-Correlation and Maximum Independence in Finite Relations

Dilian Gurov
KTH Royal Institute of Technology, Stockholm, Sweden
dilian@csc.kth.se

Minko Markov
"St. Kliment Ohridski" University of Sofia, Sofia, Bulgaria
minkom@fmi.uni-sofia.bg

We consider relations with no order on their attributes as in Database Theory. An independent partition of the set of attributes S of a finite relation R is any partition $\mathfrak{X}$ of S such that the join of the projections of R over the elements of $\mathfrak{X}$ yields R. Identifying independent partitions has many applications and corresponds conceptually to revealing orthogonality between sets of dimensions in multidimensional point spaces. A subset of S is termed self-correlated if there is a value of each of its attributes such that no tuple of R contains all those values. This paper uncovers a connection between independence and self-correlation, showing that the maximum independent partition is the least fixed point of a certain inflationary transformer α that operates on the finite lattice of partitions of S. α is defined via the minimal self-correlated subsets of S. We use some additional properties of α to show the said fixed point is still the limit of the standard approximation sequence, just as in Kleene's well-known fixed point theorem for continuous functions.

1 Introduction

The problem of discovering independence between sets of points in a multidimensional space is a fundamental problem in science. It arises naturally in many areas of Computer Science. For instance, with respect to relational data, discovering such independence allows exponential gains in storage space and processing of information [11], [1], and can facilitate the problem of machine learning [13]. With respect to problem clusterisation of multidimensional relational data, finding independence helps finding the desired clusters [5], [8]. Decomposing data into smaller units that are independent except at their interfaces has been known to be essential for understanding large legacy systems [17]. Independence has also been the subject of recent works in logic, giving rise to so-called logics of dependence and independence [4].

The concrete motivation for the present work derives from the area of *software product line engineering*, a discipline that aims at planning for and developing a *family* of products through managed reuse in order to decrease time to market and improve software quality [12]. A software family can be modelled as a relation whose attributes are the software's functionalities. The various implementations of each functionality in the form of software artefacts are the attributes' *values*. The individual products of a family are thus modelled as the tuples of that relation over the attributes. In previous works [6, 15] we considered a restricted class of software families called *simple families* (later on we changed the term "families" to the more abstract term "relations"), where discovery of independence and a compositional model checking technique are utilised to derive a *divide-and-conquer verification strategy*. Simple relations constitute the least class that contains the single-attribute, single-value relations and is closed under join of relations with disjoint attribute sets and unions of relations over the same set of attribute names but with disjoint value sets. In the present work we generalise these previous results to discovering independence in arbitrary relations. We investigate decompositions of a relation R with disjoint attributes such that R equals the join of the component relations. Every decomposition is represented by a partition of the set of attributes of R. Such partitions are termed *independent partitions*.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 60–74, doi:10.4204/EPTCS.191.7

The problem of computing a maximum decomposition of this kind has previously been studied in [10], where it is referred to as *prime factorisation*, and an efficient algorithmic solution is proposed. In this paper we investigate an alternative approach that works purely on the level of the attributes of R and is based on the concept of *correlation* between attributes. We have discovered a nontrivial connection between independence and correlation and the major goal of this paper is to demonstrate that connection.

A first observation is that the decomposition problem cannot be solved purely based on analysis of pairs of attributes. In the aforementioned work [6] we compute dependence (or independence) in simple relations by computing correlation between pairs of attributes. That approach does not generalise for arbitrary relations as we show in this paper. Our solution is to introduce *self-correlation* of sets (of arbitrary cardinality) of attributes. In other words, the current notion of correlation is a hypergraph whose hyperedges are the self-correlated sets, rather than an ordinary graph as were the case with the simple relations. Since self-correlated sets are upward closed under set inclusion (Proposition 2), the minimal self-correlated sets, or the *mincors* (Definition 4), are the foundation of our analysis. A second observation is that mincors do not cross independent partitions (Lemma 5), hence one can safely merge overlapping mincors to compute the maximum independent partition. In the case of simple relations that merger indeed yields the maximum independent partition [6] but in arbitrary relations merging the mincors *does not* necessarily output an independent partition, as the example on page 67 shows. We overcome this hindrance with the help of a final important insight. Let $\mathfrak{X}$ be the partition of the set of attributes that results from merging overlapping mincors. The relation can be factored on $\mathfrak{X}$, producing a *quotient relation*. In other words, the elements of $\mathfrak{X}$ are considered atomic now; the subsets of $\mathfrak{X}$ may or may not be self-correlated in their turn, and the said quotient relation is defined via those new mincors. We show that the procedure of identifying mincors and merging overlapping ones can be repeated on this quotient relation and this can be iterated until stabilisation, yielding the desired maximum independent partition.

The above insights suggest that relational decomposition can be presented in terms of a transformer over the finite lattice of quotient relations, or conceptually even simpler, over *the lattice of the partitions* ordered by refinement, inducing the former lattice. The transformer α on partitions introduced here essentially corresponds to identifying the mincors of the quotient relation induced by a partition, merging the overlapping ones, and extracting from the result the corresponding partition (Definition 5). We prove that the independent partitions correspond exactly to the fixed points of α (Theorem 1).

If α is monotone, one can utilise two well-known fixed point theorems on complete lattices (having in mind that monotone functions over finite lattices are continuous). First, by Tarski's fixed point theorem for complete lattices [16], the set of fixed points forms a lattice itself with respect to the same ordering, hence there is a unique *least fixed point* (LFP), which in our case would be precisely the maximum independent partitioning that we are after. And second, one can utilise Kleene's fixed point theorem [7], to the effect that the LFP can be computed *iteratively*, starting from the bottom of the lattice, *i.e.* the partition into singletons, and applying α until stabilisation, *i.e.*, until the fixed point is reached. It turns out, however, that α in general is *not monotone* as demonstrated by the example on page 70 and therefore the above reasoning is not applicable.

On the other hand, we show that α is *inflationary* (Proposition 4). The existence of a LFP is established by showing that there exists a fixed point and the set of all fixed points is closed under intersection (Lemma 6). Furthermore, the downward closure of LFP, *i.e.*, the set of all partitions refining it, is closed under α (Lemma 8). Since the lattice is finite, these results give rise to a modified version of Kleene's fixed point theorem—formulated in terms of inflationary transformers rather than monotone ones (Theorem 2)—justifying the same iterative fixed point computation procedure (Corollary 3). The proposed characterisation reduces relational decomposition to the problem of identifying the mincors of a relation.

Organisation The paper is organised as follows. Section 2 recalls some known notions and results about sets and families, partitions, lattices, fixed points, relations, attributes, and relation schemes, quotient relations, and defines independent partitions of the attributes set. Section 3 develops the theory of self-correlated sets in quotient relations and how they relate w.r.t. partition abstraction. Section 4 presents many useful lemmas that concern independence. Section 5 defines the transformer α and contains our main result, Theorem 2. Section 6 discusses what we currently know about the area of decomposition of relations, also called factorisation of relations, and compares the approach and the results of this paper with similar works. The final Section 7 draws some conclusions and outlines directions for future work.

2 Background

In this section we recall some standard set-theoretical notions and notation needed for our theoretical developments.

2.1 Sets, covers, and partitions

In this work we consider only finite sets. The powerset of a set A is denoted by $\mathtt{POW}(A)$ and $\mathtt{P}^+(A)$ denotes $\mathtt{POW}(A) \setminus \{\emptyset\}$. *Ground sets* are nonempty sets over which we construct the families that are our subject of research.

Let A be a ground set. *A family over A* is any nonempty subset of $\mathtt{P}^+(A)$. A family F is *Sperner family* if $\forall X, Y \in \mathrm{F} : X \not\subseteq Y$. F is *connected* if $\forall X, Z \in \mathrm{F}$: $X \cap Z \neq \emptyset$ or F has elements $Y_1, Y_2, \ldots, Y_k$ for some $k \geq 1$, such that $X \cap Y_1 \neq \emptyset$, $Y_i \cap Y_{i+1} \neq \emptyset$ for $1 \leq i \leq k-1$, and $Y_k \cap Z \neq \emptyset$. *A connected component of a family* is any maximal connected subfamily in it. We use $\mathtt{CC}(\mathrm{F})$ to denote the family $\{\cup \mathrm{B} \,|\, \mathrm{B} \text{ is a connected component of F}\}$. *A superfamily over A* is any nonempty subset of $\mathtt{P}^+(\mathtt{P}^+(A))$.

Suppose A is a set. *A cover of A* is any family F over A such that $\cup \mathrm{F} = A$. The set of all covers of A is denoted by $K(A)$. If $\mathfrak{X} \in K(A)$ and $Y \cap Z = \emptyset$ for all distinct $Y, Z \in \mathfrak{X}$, we say $\mathfrak{X}$ is *a partition of A*. If $|\mathfrak{X}| = 1$ the partition is *trivial* and if $|\mathfrak{X}| = |A|$ the partition is *partition into singletons*. Note that $\mathtt{CC}(\mathrm{F})$ defined above is a partition of the ground set. We denote by $\mathfrak{Y} \Subset \mathfrak{X}$ the fact that for some $B \subseteq A$, $\mathfrak{Y}$ is a family over B such that every element of $\mathfrak{Y}$ is a subset of precisely one element of $\mathfrak{X}$ and every element of $\mathfrak{X}$ is a superset of at most one element of $\mathfrak{Y}$. For example, if $A = \{a, b, c, d, e, f, g, h, k\}$ then $\{\{b\}, \{c\}, \{d, g\}\} \Subset \{\{a, b\}, \{c\}, \{d, e, f, g\}, \{h, k\}\}$.

The set of all partitions of A is denoted by $\Pi(A)$. For any $P_1, P_2 \in \Pi(A)$, P_1 *refines* P_2, which we denote by $P_1 \sqsubseteq P_2$, if

$$\forall X \in P_1 \;\; \exists Y \in P_2 : X \subseteq Y$$

Conversely, we say that P_2 *abstracts* P_1. If $P_1 \sqsubseteq P_2$ and $P_1 \neq P_2$ we write $P_1 \sqsubset P_2$.

2.2 Partial orders, lattices, and chains

We denote generic partial orders by "$\preccurlyeq$". If $(A, \preccurlyeq)$ is a poset, *a least element* of A is any $x \in A$ such that $\forall y \in A : x \preccurlyeq y$ and *a greatest element* of A is any $x \in A$ such that $\forall y \in A : y \preccurlyeq x$. A least element may not exist but if it exists it is unique; the same holds for a greatest element. The least element is called *bottom* and is denoted by $\bot$. The greatest element is called *top* and is denoted by $\top$. *A chain* in a poset $(A, \preccurlyeq)$ is any $B \subseteq A$ such that $\forall x, y \in B : x \preccurlyeq y \vee y \preccurlyeq x$.

A lattice is a poset $(A, \preccurlyeq)$, shortly A when $\preccurlyeq$ is understood, such that for any $x, y \in A$ there exists a (unique) greatest lower bound in A called *meet* and denoted by $x \sqcap y$ and a (unique) least upper bound

in A called *join* and denoted by $x \sqcup y$. Collectively, $\sqcap$ and $\sqcup$ are *the lattice operations of A*. They are commutative and associative [2, pp. 8]. We generalise the lattice operations on subsets of A in the obvious way. *A complete lattice* is a lattice such that every $B \subseteq A$ has a meet $\sqcap B$ and a join $\sqcup B$. In particular, A has a meet $\sqcap A = \bot$ and a join $\sqcup A = \top$. Every finite lattice is complete [3, pp. 46], therefore from now on by lattice we mean complete lattice. For any $x \in A$, the sets $\{y \in A \,|\, y \preccurlyeq x\}$ and $\{y \in A \,|\, x \preccurlyeq y\}$ are called *down-x* and *up-x* and are denoted by $\uparrow x$ and $\downarrow x$, respectively [3, pp. 20].

It is well-known that $(\Pi(A), \sqsubseteq)$ is a lattice. Furthermore, $\bot$ is the partition into singletons, $\top$ is the trivial partition, and for any $P_1, P_2 \in \Pi(A)$, $P_1 \sqcap P_2 = \{X \cap Y \,|\, X \in P_1, Y \in P_2\} \setminus \{\emptyset\}$ and $P_1 \sqcup P_2 = \mathcal{C}(P_1 \cup P_2)$ (see [2, pp. 15]). We extend the "$\sqcap$" notation to subsets of partitions: for any $\mathfrak{X}, \mathfrak{Y} \in \Pi(A)$, for any nonempty $\mathfrak{X}' \subseteq \mathfrak{X}$ and any nonempty $\mathfrak{Y}' \subseteq \mathfrak{Y}$ such that $\mathfrak{X}' \cap \mathfrak{Y}' \neq \emptyset$, $\mathfrak{X}' \sqcap \mathfrak{Y}'$ denotes the set $\{B \cap C \,|\, B \in \mathfrak{X}', C \in \mathfrak{Y}'\} \setminus \{\emptyset\}$.

2.3 Functions and fixed points

Suppose A is a set and $f : A \to A$ is a function. For every $x \in A$: $f^0(x) \stackrel{\text{def}}{=} x$ and for every $n \in \mathbb{N}^+$, $f^n(x) \stackrel{\text{def}}{=} f \circ f^{n-1}(x)$. For every $n \in \mathbb{N}$, $f^n(x)$ is *the n-th iterate of f*. *A fixed point* of f is every $x \in A$ such that $f(x) = x$. Let $(A, \preccurlyeq)$ be a poset. A function $f : A \to A$ is *monotone* if $\forall x, y \in A : x \preccurlyeq y \to f(x) \preccurlyeq f(y)$ and f is *inflationary* if $\forall x \in A : x \preccurlyeq f(x)$ [14, pp. 263].

A well-known fixed point theorem is Tarski's fixed point theorem for continuous functions over complete lattices [16], stating that the set of fixed points is non-empty and forms a lattice itself with respect to the same ordering, and hence the function has a unique *least fixed point* (LFP). Another well-known theorem due to Kleene states the existence of an LFP for continuous functions on chain-complete partial orders [7], and that the LFP can be computed *iteratively*, starting from the bottom of the lattice and applying the function until stabilisation.

2.4 Schemes, relations, and quotient relations

The following definitions are close to the ones in [9]. *A scheme* is a nonempty set $\mathrm{S} = \{A_1, \ldots, A_n\}$ whose elements, called *the attributes*, are nonempty sets. For every attribute, its elements are said to be its *values*. *A relation over S* is a nonempty set of total functions $\{t_1, t_2, \ldots, t_p\}$, which we call *the tuples*, such that for $1 \leq j \leq p$, $t_j : \mathrm{S} \to \cup\mathrm{S}$, with the restriction that $t_j(A_i) \in A_i$, for $1 \leq i \leq n$. We assume that every value of every attribute occurs in at least one tuple.

The relations we have in mind are as in Relational Database Theory, *i.e.* with unordered tuples, rather than as in Set Theory, *i.e.* with ordered tuples.

We further postulate that the said attributes are mutually disjoint sets. That allows a simplification of the definition of relation: a relation over S is nonempty set of tuples, each tuple being an n-element set with precisely one element from every attribute. To save space, we often write the tuples without commas between their elements. For example, let $n = 3$, $A_1 = \{a_1, a_2\}$, $A_2 = \{b_1, b_2\}$, and $A_3 = \{c_1, c_2, c_3\}$. One of the relations over the scheme $\{A_1, A_2, A_3\}$ is written as $\{\{a_1b_1c_1\}, \{a_1b_2c_2\}, \{a_2b_2c_3\}\}$.

Let $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_k$ be schemes such that for $1 \leq i < j \leq k$, $\forall A \in \mathrm{S}_i \; \forall B \in \mathrm{S}_j : A \cap B = \emptyset$. Let R_i be a relation over S_i, for $1 \leq i \leq k$. *The join of* $R_1, \ldots, R_k$ is the relation

$$R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k = \{\cup\{x_1, x_2, \ldots, x_k\} \,|\, x_1 \in R_1, x_2 \in R_2, \ldots, x_k \in R_k\}$$

The complete relation over $\mathrm{S} = \{A_1, \ldots, A_n\}$ is $\bowtie_{i=1}^{n} \{\{x\} \,|\, x \in A_i\}$. Clearly, its cardinality is $\prod_{i=1}^{n} |A_i|$.

Let $\mathrm{S} = \{A_1, \ldots, A_n\}$ be a scheme. *A subscheme of* S is any nonempty subset of S. The notation $f\big|_Z$ stands for the restriction of f to Z, for any function $f : X \to Y$ and any $Z \subseteq X$. Let $R = \{t_1, t_2, \ldots, t_p\}$ be a relation over S and let T be a subscheme of S. *The projection of R on* T is $R \restriction \mathrm{T} = \{t_j\big|_{\mathrm{T}} : 1 \leq j \leq p\}$.

Definition 1 (quotient relation) *Let R be a relation over some scheme* S. *For any* $\mathfrak{X} = \{\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_n\} \in \Pi(\mathrm{S})$, $R/\mathfrak{X} \subseteq \bowtie_{i=1}^{n} (R \restriction \mathrm{X}_i)$ *is the following relation:*

$$\forall \{y_1 y_2 \ldots y_n\} \in \bowtie_{i=1}^{n} (R \restriction \mathrm{X}_i) : \\ \{y_1 y_2 \ldots y_n\} \in R/\mathfrak{X} \text{ iff } \exists t \in R \, \forall i_{1 \leq i \leq n} (t \restriction \mathrm{X}_i = y_i)$$

We term $R/\mathfrak{X}$ *the* quotient relation of R relative to $\mathfrak{X}$. *When* $\mathfrak{X}$ *is understood we say simply* the quotient relation of R.

We emphasise the quotient relation is not over S but over a partition of S.

Here is an example of a quotient relation. Let $\mathrm{S} = \{A, B, C, D\}$, let each attribute have precisely two values, say $A = \{a_1, a_2\}$ and so on, let $\mathfrak{X}_1 = \{\{A, B\}, \{C, D\}\}$, let $\mathfrak{X}_2 = \{\{A\}, \{B\}, \{C\}, \{D\}\}$, and let

$$R' = \{\{a_1 b_1 c_1 d_1\}, \{a_1 b_1 c_2 d_2\}, \{a_1 b_2 c_1 d_2\}, \{a_2 b_2 c_1 d_1\}, \{a_2 b_2 c_2 d_2\}\} \tag{1}$$

be a relation over S. Then

$$R'/\mathfrak{X}_1 = \{\{\{a_1, b_1\}\{c_1, d_1\}\}, \{\{a_1, b_1\}\{c_2, d_2\}\}, \{\{a_1, b_2\}\{c_1, d_2\}\}, \\ \{\{a_2, b_2\}\{c_1, d_1\}\}, \{\{a_2, b_2\}\{c_2, d_2\}\}\} \tag{2}$$

$$R'/\mathfrak{X}_2 = \{\{\{a_1\}\{b_1\}\{c_1\}\{d_1\}\}, \{\{a_1\}\{b_1\}\{c_2\}\{d_2\}\}, \{\{a_1\}\{b_2\}\{c_1\}\{d_2\}\}, \\ \{\{a_2\}\{b_2\}\{c_1\}\{d_1\}\}, \{\{a_2\}\{b_2\}\{c_2\}\{d_2\}\}\} \tag{3}$$

A quotient relation is but a grouping together of the tuples of the original relation into subtuples according to the partition. It trivially follows that $|R/\mathfrak{X}| = |R|$ for any relation R over any attribute set S and any $\mathfrak{X} \in \Pi(\mathrm{S})$.

2.5 Independent partitions

For a given relation R over some scheme S, we are after decompositions of R such that R equals the join of the obtained components. Each decomposition of this kind corresponds to a certain partition of S.

Definition 2 (independent partition) *Let R be a relation over some scheme S. For any* $\mathfrak{X} \in \Pi(\mathrm{S})$, $\mathfrak{X}$ *is* an independent partition of S with respect to R *if* $R = \bowtie_{Y \in \mathfrak{X}} R \restriction Y$. *The set of all independent partitions of* S *with respect to R is denoted by* $\mathrm{I}\Pi_R(\mathrm{S})$, *or shortly* $\mathrm{I}\Pi(\mathrm{S})$ *if R is understood. If a partition is not independent, it is* dependent.

Note that $\mathrm{I}\Pi(\mathrm{S})$ is nonempty since it necessarily contains the trivial partition.

Proposition 1 *For every independent partition* $\mathfrak{X}$, $R/\mathfrak{X}$ *is the complete relation over* $\mathfrak{X}$.

Informally speaking, the object of the present study is the independent partition with the maximum number of equivalence classes, provided it is unique.

3 Correlation in Relations

In this section we define correlation in relations and quotient relations. From now on assume an arbitrary but fixed scheme S and relation R over it.

3.1 Correlated subsets of ground sets

In this subsection, the ground sets are schemes.

Definition 3 (correlated subsets of schemes) *Let* $\mathrm{S} = \{A_1, A_2, \ldots, A_n\}$ *and let* T *be some nonempty subscheme* $\{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}$ *where* $1 \leq i_1 < i_2 < \cdots < i_m \leq n$. T *is* self-correlated with respect to *R, or shortly* correlated with respect to *R, iff*

$$\exists x_1 \in A_{i_1}\ \exists x_2 \in A_{i_2}\ \cdots\ \exists x_m \in A_{i_m} : \{x_1 x_2 \cdots x_m\} \notin R \upharpoonright \mathrm{T} \tag{4}$$

We denote that fact by $corr_R(\mathrm{T})$ *or* $corr(\mathrm{T})$ *if R is understood. The opposite concept is* uncorrelated*. The family* $\{\mathrm{T} \subseteq \mathrm{A} \,|\, corr_R(\mathrm{T})\}$*, in case it is nonempty, is called* the correlation family of *R.*

Note that no minimal correlated subset is a singleton. The following result re-states correlation of a subscheme in terms of the projection of the relation on it.

Lemma 1 *Let* $\mathrm{T} \subseteq \mathrm{S}$. *Then* $corr(\mathrm{T})$ *iff* $R \upharpoonright \mathrm{T} \subsetneq \bowtie_{X \in \mathrm{T}} R \upharpoonright \{X\}$.

Proof: First assume corr(T). By Definition 3, there is an element in every attribute from T such that the tuple of those elements does not occur in $R \upharpoonright \mathrm{T}$. On the other hand, the tuples of $\bowtie_{X \in \mathrm{T}} R \upharpoonright \{X\}$ are all possible combinations of the elements of the attributes in T. Therefore, $R \upharpoonright \mathrm{T} \subsetneq \bowtie_{X \in \mathrm{T}} R \upharpoonright \{X\}$.

In the other direction, assume ¬corr(T). The negation of expression (4) in Definition 3 is but another way to write $R \upharpoonright \mathrm{T} = \bowtie_{X \in \mathrm{T}} R \upharpoonright \{X\}$. □

As the next result establishes, with respect to the poset $(\mathrm{S}, \subseteq)$, every correlated subset is upward closed, while every uncorrelated subset is downward closed.

Proposition 2 *If* $corr(\mathrm{T})$ *for some* $\mathrm{T} \subseteq \mathrm{S}$ *then* $\forall \mathrm{Z}_{\mathrm{T} \subseteq \mathrm{Z} \subseteq \mathrm{S}} : corr(\mathrm{Z})$. *If* $\neg corr(\mathrm{T})$ *for some* $\mathrm{T} \subseteq \mathrm{S}$ *then* $\forall \mathrm{Z}_{\mathrm{Z} \subseteq \mathrm{T}} : \neg corr(\mathrm{Z})$.

It is obvious that the correlation family, if it exists, is a cover of the scheme. Furthermore, it does not exist iff the relation is complete. The interesting part of a correlation family is the sub-family comprising the minimal correlated sets. However, that sub-family does not necessarily cover the scheme. We want to define a family that both covers the scheme—because we are ultimately interested in a partition of the scheme—and is a Sperner family, since the implied members of the family are of no interest.

Definition 4 (mincor family) A mincor of *R is every minimal, self-correlated with respect to R, subscheme* $\mathrm{T} \subseteq \mathrm{S}$. *Further,* $\mathsf{mincors}(R) \stackrel{def}{=} \{\mathrm{T} \subseteq \mathrm{S} \,|\, \mathrm{T}$ *is a mincor*$\}$ *and* $\mathsf{singletons}(R) \stackrel{def}{=} \{\{A\} \,|\, A \in \mathrm{S} \wedge \neg\exists X \in \mathsf{mincors}(R) : A \in X\}$. The mincor family *of R, denoted by* $\mathsf{MF}(R)$*, is* $\mathsf{MF}(R) = \mathsf{mincors}(R) \cup \mathsf{singletons}(R)$.

For example, consider R' defined in (1) on the facing page. Clearly, $\mathrm{corr}_{R'}(\{A,B\})$ and $\mathrm{corr}_{R'}(\{C,D\})$ because of the lacks of both a_2 and b_1 in any tuple and the lack of both c_2 and d_1 in any tuple, respectively. The other four two-element subsets of S are uncorrelated. Then $\mathsf{singletons}(R') = \emptyset$ and therefore $\mathsf{MF}(R') = \{\{A,B\},\{C,D\}\}$.

Proposition 3 *With respect to* S *and R,* $\mathsf{MF}(R)$ *exists and is unique.*

If *R* is complete then $\mathsf{MF}(R)$ consists of singletons. Clearly, $\mathsf{MF}(R) \in K(S)$, and thus $\mathsf{CC}(\mathsf{MF}(R)) \in \Pi(S)$.

3.2 Correlation in quotient relations

The following result establishes an important connection between self-correlation in a partition of the scheme and self-correlation in the scheme itself. More specifically, Lemma 2 is used to prove Lemma 3, and the latter is used in the proof of Lemma 7 on page 71.

Lemma 2 *For any* $\mathfrak{X} \in \Pi(\mathrm{S})$ *and* $\mathfrak{X}' \subseteq \mathfrak{X}$*:*

$$corr_{R/\mathfrak{X}}(\mathfrak{X}') \leftrightarrow corr_R(\cup\mathfrak{X}')$$

Proof: Assume $\mathrm{corr}_{R/\mathfrak{X}}(\mathfrak{X}')$. Let $\mathfrak{X}' = \{Y_1, Y_2, \ldots, Y_m\}$. So, $(R/\mathfrak{X}) \restriction \mathfrak{X}'$ does not contain some m-tuple $\{U_1, U_2, \ldots, U_m\}$ such that $U_i \in R \restriction Y_i$ for $1 \leq i \leq m$. Then $R \restriction \cup\mathfrak{X}'$ does not contain $\cup\{U_1, U_2, \ldots, U_m\}$.

In the other direction, assume $\mathrm{corr}_R(\cup\mathfrak{X}')$ where $\cup\mathfrak{X}'$ is a subset S′ of S. Let $\mathrm{S}' = \{A_1, A_2, \ldots, A_n\}$. That is, $R \restriction \mathrm{S}'$ does not contain some n-tuple $\{W_1, W_2, \ldots, W_n\}$ such that $W_i \in A_i$ for $1 \leq i \leq n$. Let $\mathfrak{X}' = \{Y_1, Y_2, \ldots, Y_m\}$. Then $(R/\mathfrak{X}) \restriction \mathfrak{X}'$ does not contain the m-tuple $\{U_1, U_2, \ldots, U_m\}$ where $U_i \in R \restriction Y_i$ for $1 \leq i \leq m$. □

As an example that illustrates Lemma 2, consider R' and $\mathfrak{X}_1$ on page 64. Clearly, $\mathfrak{X}_1 = \{\{A, B\}, \{C, D\}\}$ is self-correlated with respect to $\tilde{R}/\mathfrak{X}_1$ as $\tilde{R}/\mathfrak{X}_1$ does not contain, among others, the tuple $\{\{a_1, b_1\}\{c_1, d_2\}\}$. That implies $\cup\mathfrak{X}_1 = \{A, B, C, D\}$ is self-correlated with respect to $\tilde{R}$: since $\{\{a_1, b_1\}\{c_1, d_2\}\}$ is not an element of $\tilde{R}/\mathfrak{X}_1$, it must be the case that $\{a_1 b_1 c_1 d_2\}$ is not element of $\tilde{R}$ (and indeed it is not). In the other direction, the fact that $\{a_1 b_1 c_1 d_2\} \notin \tilde{R}$ implies $\{\{a_1, b_1\}\{c_1, d_2\}\} \notin \tilde{R}/\mathfrak{X}_1$.

The next result establishes that for every mincor Y of a quotient relation there is a way to pick elements from every element of Y such that the collection of those elements is a mincor of the original relation R.

Lemma 3 $\forall\mathfrak{X} \in \Pi(\mathrm{S})\ \forall\mathfrak{Y} \in \mathsf{mincors}(R/\mathfrak{X})\ \exists \mathrm{Z} \Subset \mathfrak{Y} : |\mathrm{Z}| = |\mathfrak{Y}| \wedge \cup\mathrm{Z} \in \mathsf{mincors}(R)$.

Proof: Assume $\mathfrak{Y} \in \mathsf{mincors}(R/\mathfrak{X})$. Clearly, there is some $\mathrm{Z} \Subset \mathfrak{Y}$ such that $\cup\mathrm{Z}$ is correlated with respect to R because $\Subset$ is reflexive and $\cup\mathfrak{Y}$ is correlated with respect to R by Lemma 2. Now consider any $\mathrm{Z}' \Subset \mathfrak{Y}$ such that $|\mathrm{Z}'| < |\mathfrak{Y}|$. There exists some $\mathfrak{Y}' \subset \mathfrak{Y}$ such that $\mathrm{Z} \Subset \mathfrak{Y}'$. But $\mathfrak{Y}'$ is uncorrelated with respect to $R/\mathfrak{X}$ because $\mathfrak{Y}$ is a mincor of $R/\mathfrak{X}$ and so every proper subset of $\mathfrak{Y}$ is uncorrelated with respect to $R/\mathfrak{X}$. Note that $\mathfrak{Y}'$ being uncorrelated with respect to $R/\mathfrak{X}$ implies $\cup\mathrm{Z}'$ is uncorrelated with respect to R by Lemma 2. It follows that for any $\mathrm{Z} \Subset \mathfrak{Y}$ such that $\mathrm{corr}_R(\cup\mathrm{Z})$—and we established such a Z exists—it is the case that $|\mathrm{Z}| = |\mathfrak{Y}|$.

So, there exists a $\mathrm{Z} \Subset \mathfrak{Y}$ such that $|\mathrm{Z}| = |\mathfrak{Y}|$ and $\cup\mathrm{Z}$ is correlated with respect to R. Furthermore, there does not exist $\mathrm{Z} \Subset \mathfrak{Y}$ such that $|\mathrm{Z}| < |\mathfrak{Y}|$ and $\cup\mathrm{Z}$ is correlated with respect to R. Consider any $\tilde{\mathrm{Z}} \Subset \mathfrak{Y}$ such that $\cup\tilde{\mathrm{Z}}$ is correlated with respect to R. As $|\tilde{\mathrm{Z}}| = |\mathfrak{Y}|$, every element of $\mathfrak{Y}$ is a superset of precisely one element of $\tilde{\mathrm{Z}}$.

First assume all elements of $\tilde{\mathrm{Z}}$ are singletons. In this case no proper subset of $\cup\tilde{\mathrm{Z}}$ is correlated with respect to R. Suppose the contrary, namely that some $\mathrm{W} \subset \cup\tilde{\mathrm{Z}}$ is correlated with respect to R and deduce there is some $\mathrm{Z}'' \Subset \mathfrak{Y}$ such that $\mathrm{W} = \cup\mathrm{Z}''$, thus $|\mathrm{Z}''| < |\mathfrak{Y}|$, such that $\cup\mathrm{Z}''$ is correlated with respect to R. Since no proper subset of $\cup\tilde{\mathrm{Z}}$ is correlated with respect to R, $\cup\tilde{\mathrm{Z}}$ is a mincor with respect to R and we are done with the proof.

Now assume not all elements of $\tilde{\mathrm{Z}}$ are singletons. It trivially follows there exists a minimal set $\widehat{\mathrm{Z}} \Subset \tilde{\mathrm{Z}}$ such that $|\widehat{\mathrm{Z}}| = |\tilde{\mathrm{Z}}|$ (thus $|\widehat{\mathrm{Z}}| = |\mathfrak{Y}|$) such that $\cup\tilde{\mathrm{Z}}$ is correlated with respect to R. □

4 Results on Independent Partitions

This section provides important auxiliary results concerning independent partitions. In subsection 4.1 we investigate the connection between independence and self-correlation. In subsection 4.2 we prove the meet of independent partitions is an independent partition.

4.1 Independence and the mincor family

The following lemma establishes that partition independence is preserved under removal of attributes.

Lemma 4 $\forall \mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S})\ \forall \mathfrak{X} \Subset \mathfrak{Y} : \mathfrak{X} \in \mathrm{I\Pi}_{R \restriction \cup \mathfrak{X}}(\cup \mathfrak{X})$.

Proof: Let $Q = R \restriction \cup \mathfrak{X}$. We prove that $Q = \bowtie_{Z \in \mathfrak{X}} (Q \restriction Z)$. In one direction, $Q \subseteq \bowtie_{Z \in \mathfrak{X}} (Q \restriction Z)$ follows immediately from the definitions of relation join and projection. In the other direction, consider any tuple t in $\bowtie_{Z \in \mathfrak{X}} (Q \restriction Z)$. Let v be any tuple in $\bowtie_{Z \in \mathfrak{Y}} (R \restriction Z)$ such that $t = v\big|_{\cup \mathfrak{X}}$. But $v \in R$ because $\mathfrak{Y}$ is independent and thus $R = \bowtie_{Z \in \mathfrak{Y}} (R \restriction Z)$. As $v \in R$, it follows that $v\big|_{\cup \mathfrak{X}} \in Q$. But $v\big|_{\cup \mathfrak{X}}$ is t, therefore $t \in Q$, and so $\bowtie_{Z \in \mathfrak{X}} (Q \restriction Z) \subseteq Q$. □

The next lemma is pivotal. It shows that the mincors respect independent partitions, in the sense that no mincor can intersect more than one element of an independent partition.

Lemma 5 $\forall \mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S})\ \ \forall \mathrm{W} \in \mathsf{mincors}(R)\ \ \exists \mathrm{Y} \in \mathfrak{Y} : \mathrm{W} \subseteq \mathrm{Y}$.

Proof: Assume the contrary. Then there is a mincor W that has nonempty intersection with more than one set from $\mathfrak{Y}$. Suppose W has nonempty intersection with precisely t sets from $\mathfrak{Y}$ for some t such that $2 \leq t \leq q$. Let $\mathrm{Y}_1, \mathrm{Y}_2, \ldots, \mathrm{Y}_t$ be precisely those sets from $\mathfrak{Y}$ that have nonempty intersection with W. Let $\mathrm{W}_i = \mathrm{W} \cap \mathrm{Y}_i$, for $1 \leq i \leq t$. Clearly, $\bigcup_{i=1}^{t} \mathrm{W}_i = \mathrm{W}$. By Lemma 4:

$$R \restriction \mathrm{W} = \bowtie_{1 \leq i \leq t} R \restriction \mathrm{W}_i$$

Every W_i is a proper subset of W. But W is a minimal correlated set. That implies $\neg\mathrm{corr}(\mathrm{W}_i)$, for $1 \leq i \leq t$. Apply Lemma 1 to conclude that $R \restriction \mathrm{W}_i = \bowtie_{x \in \mathrm{W}_i} R \restriction \{x\}$. Then,

$$R \restriction \mathrm{W} = \bowtie_{1 \leq i \leq t} \bowtie_{x \in \mathrm{W}_i} R \restriction \{x\}$$

Obviously, $\bowtie_{1 \leq i \leq t} \bowtie_{x \in \mathrm{W}_i} R \restriction \{x\} = \bowtie_{x \in \mathrm{W}} R \restriction \{x\}$. Then, $R \restriction \mathrm{W} = \bowtie_{x \in \mathrm{W}} R \restriction \{x\}$. By Lemma 1 that implies $\neg\mathrm{corr}(\mathrm{W})$. □

Furthermore, merging mincors also yields sets that respect independent partitions.

Corollary 1 $\forall \mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S}) : \mathtt{CC}(\mathtt{MF}(R)) \sqsubseteq \mathfrak{Y}$.

Proof: Assume the contrary. Then for some R on S and $\mathfrak{Y} \in \mathrm{I\Pi}(S)$:

$$\exists \mathrm{X} \in \mathtt{CC}(\mathtt{MF}(R))\ \ \forall \mathrm{Y} \in \mathfrak{Y}\ \ \exists A \in \mathrm{X} : A \notin \mathrm{Y}$$

First note that X is not a singleton, otherwise X would be contained in some set from $\mathfrak{Y}$. So, $|\mathrm{X}| \geq 2$ and according to Definition 4, X is the union of one or more mincors, each of size ≥ 2, and X is connected. But by assumption X is not a subset of any set from $\mathfrak{Y}$ and so there has to be some mincor $\mathrm{W} \in \mathrm{X}$ that has nonempty intersection with at least two sets from $\mathfrak{Y}$. However, that contradicts Lemma 5. □

Note that $\mathtt{CC}(\mathtt{MF}(R))$ is not necessarily an independent partition. For example, consider R' defined in (1) on page 64. As explained on page 65, $\mathtt{MF}(R') = \{\{A,B\},\{C,D\}\}$ and thus $\mathtt{CC}(\mathtt{MF}(R')) = \{\{A,B\},\{C,D\}\}$, too. But $\{\{A,B\},\{C,D\}\}$ is not an independent partition with respect to R'. In fact, there is no independent partition of S except for the trivial partition as $|R'|$ is a prime number.

Now consider another relation R'' on the same scheme:

$$R'' = \{\{a_1b_1c_1d_1\},\{a_1b_1c_1d_2\},\{a_1b_1c_2d_2\},\{a_1b_2c_1d_1\},\{a_1b_2c_1d_2\},\{a_1b_2c_2d_2\},\\ \{a_2b_2c_1d_1\},\{a_2b_2c_1d_2\},\{a_2b_2c_2d_2\}\}$$

But $\mathtt{MF}(R'') = \{\{A,B\},\{C,D\}\} = \mathtt{CC}(\mathtt{MF}(R''))$ just as in the case of R'. Now $\{\{A,B\},\{C,D\}\}$ is an independent partition with respect to R'' because $R'' = R'' \restriction \{A,B\} \bowtie R'' \restriction \{C,D\}$.

So, in the case of R'', the connected components of the mincor family constitute an independent partition, while that is not true for R', although the mincor families of both relations are the same. We conclude that computing the mincor family does not suffice to obtain an independent partition. Therefore, we use a more involved approach in which the computation of the mincor family is but the first step towards the computation of the maximum independent partition.

4.2 The meet of independent partitions

The following lemma allows us to define the maximum independent partition as the meet of all independent partitions.

Lemma 6 $\forall \mathfrak{X},\mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S}) : \mathfrak{X} \sqcap \mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S})$.

Proof: (sketch) Let $\mathfrak{X},\mathfrak{Y} \in \mathrm{I\Pi}(\mathrm{S})$. We assume $\mathfrak{X} \sqcup \mathfrak{Y}$ is connected. There is no true loss of generality in that because the proof below can be done componentwise if $\mathfrak{X} \sqcup \mathfrak{Y}$ is not connected. Relative to an arbitrary element of $\mathfrak{X}$, say X_1, we define the family $\mathfrak{Z} = \{Z_0, Z_1, \ldots, Z_k\}$ over S as follows. $\mathfrak{Z}$ is a partition of S and its elements are constructed in an ascending order of the index according to the following rule:

$$Z_i = \begin{cases} X_1, & \text{if } i = 0 \\ \bigcup\{A \setminus Z_{i-1} \,|\, A \in \mathfrak{Y} \wedge A \cap Z_{i-1} \neq \emptyset\}, & \text{if } i \text{ is odd} \\ \bigcup\{A \setminus Z_{i-1} \,|\, A \in \mathfrak{X} \wedge A \cap Z_{i-1} \neq \emptyset\}, & \text{if } i \text{ is even and } i > 0 \end{cases}$$

Let us define $B_i = \{\bigcup_{j=0}^{i} Z_j\} \sqcap \mathfrak{X} \sqcap \mathfrak{Y}$ for $0 \leq i \leq k$. Clearly, $B_0 = \{X_1\} \sqcap \mathfrak{Y}$, $B_i = B_{i-1} \cup (\{Z_i\} \sqcap \mathfrak{X} \sqcap \mathfrak{Y})$ for $1 \leq i \leq k$ and $B_k = \mathfrak{X} \sqcap \mathfrak{Y}$. Furthermore, $\cup B_k = \mathrm{S}$ and thus $R \restriction \cup B_k = R$. We prove by induction on i that for all i such that $0 \leq i \leq k$:

$$R \restriction \cup B_i = \underset{C \in B_i}{\bowtie} R \restriction C \tag{5}$$

and hence the result follows.

Basis. Let $i = 0$. Let the elements of $\mathfrak{Y}$ that have nonempty intersection with X_1 be called $Y_1, \ldots, Y_j$. Obviously, there is at least one of them. The claim is that $R \restriction X_1 = \bowtie_{i=1}^{j} R \restriction (X_1 \cap Y_i)$. That follows immediately from Lemma 4.

Inductive Step. Assume the claim holds for some B_{i-1} such that $0 \leq i-1 < k$ and consider B_i. As already mentioned, $B_i = B_{i-1} \cup (\{Z_i\} \sqcap \mathfrak{X} \sqcap \mathfrak{Y})$.

Without loss of generality, assume i is odd. Very informally speaking, Z_i is the union of some elements of $\mathfrak{Y}$ that overlap with some elements (from $\mathfrak{X}$) in B_{i-1}, minus the overlap. Therefore, we can write $B_i = B_{i-1} \cup (\{Z_i\} \sqcap \mathfrak{X})$ because under the current assumption, it is $\mathfrak{X}$ rather than $\mathfrak{Y}$ that dictates the grouping together of the elements of Z_i in B_i. More specifically, since $i \neq k$, there are elements from $\mathfrak{X}$ whose elements do not appear in the current B_i; those elements of $\mathfrak{X}$ dictate the aforementioned grouping.

So, B_i is the union of two disjoint sets whose elements are from $\mathfrak{X} \sqcap \mathfrak{Y}$, namely B_{i-1} and $\{Z_i\} \sqcap \mathfrak{X}$. By the inductive hypothesis, $R \restriction \cup B_{i-1} = \bowtie_{C \in B_{i-1}} R \restriction C$.

Consider $\{Z_i\} \sqcap \mathfrak{X}$ and call its elements, $T_1, \ldots, T_m$. Without loss of generality, consider T_1. Our immediate goal is to prove that $R \restriction ((\cup B_{i-1}) \cup T_1) = \bowtie_{C \in B_{i-1} \cup \{T_1\}} R \restriction C$. Note that T_1 is a subset of some $Y' \in \mathfrak{Y}$ such that Y' has nonempty intersection with $\cup B_{i-1}$, T_1 itself being disjoint with B_{i-1}. Furthermore, T_1 is the intersection of Y' with some $X' \in \mathfrak{X}$. X' is disjoint with $\cup B_{i-1}$, otherwise the elements of T_1 would be part of $\cup B_{i-1}$. Furthermore, every element of B_{i-1} is a subset of some element of $\mathfrak{X}$ that is not X'. Let the elements of $\mathfrak{X}$ that have subsets-elements of B_{i-1} be $X_1, \ldots, X_p$. Note that $X_1 \cup \cdots \cup X_p = \cup B_{i-1}$. By Lemma 4, it is the case that

$$R \restriction (X_1 \cup \cdots \cup X_p \cup T_1) = R \restriction X_1 \bowtie \cdots \bowtie R \restriction X_p \bowtie R \restriction T_1 \tag{6}$$

since T_1 is a subset of X' and X' is none of $X_1, \ldots, X_p$. However, $X_1 \cup \cdots \cup X_p \cup T_1 = (\cup B_{i-1}) \cup T_1$ by an earlier observation and $R \restriction X_1 \bowtie \cdots \bowtie R \restriction X_p = \bowtie_{C \in B_{i-1}} R \restriction C$. Substitute that in equation 6 to obtain

$$R \restriction (\cup B_{i-1} \cup T_1) = \left(\bowtie_{C \in B_{i-1}} R \restriction C \right) \bowtie R \restriction T_1 = \bowtie_{C \in B_{i-1} \cup \{T_1\}} R \restriction C \tag{7}$$

which is what we wanted to prove with respect to T_1.

We can use (7) as the basis of a nested induction. More specifically, we prove that

$$R \restriction ((\cup B_{i-1}) \cup T_1 \cup \cdots \cup T_k) = \left(\bowtie_{C \in B_{i-1}} R \restriction C \right) \bowtie R \restriction T_1 \bowtie \cdots \bowtie R \restriction T_k$$

implies

$$R \restriction ((\cup B_{i-1}) \cup T_1 \cup \cdots \cup T_{k+1}) = \left(\bowtie_{C \in B_{i-1}} R \restriction C \right) \bowtie R \restriction T_1 \bowtie \cdots \bowtie R \restriction T_{k+1}$$

for any $k \in \{1, 2, \ldots, m-1\}$. The nested induction can be proved in a straightforward manner, having in mind the proof of (7). That implies the desired:

$$R \restriction ((\cup B_{i-1}) \cup T_1 \cup \cdots \cup T_m) = \left(\bowtie_{C \in B_{i-1}} R \restriction C \right) \bowtie R \restriction T_1 \bowtie \cdots \bowtie R \restriction T_m$$

And that concludes the proof because $\cup B_i = \cup B_{i-1} \cup T_1 \cup \cdots \cup T_m$. □

The proof of Lemma 6 relies on the fact that all sets we consider are finite.

As a corollary of Lemma 6, the maximum independent partition, which is the object of our study, is well-defined: $\sqcap \mathrm{I}\Pi(\mathrm{S})$ exists, it is unique, and is an element of $\mathrm{I}\Pi(\mathrm{S})$. For notational convenience we introduce another term for that object. We say that $\sqcap \mathrm{I}\Pi_R(\mathrm{S})$ is the *focus* of R and denote it by $\mathsf{foc}(R)$. A trivial observation is that $\mathrm{I}\Pi_R(\mathrm{S})$ coincides with $\uparrow \mathsf{foc}(R)$.

5 A Fixed Point Characterisation of the Maximum Independent Partition

In this section we identify the object of our study as the least fixed point of α, where α is a transformer on the lattice of all partitions of S. Furthermore, we present an iterative fixed point approximation procedure for computing the maximum independent partition.

5.1 Function α

First we introduce a helper function. Let A be a ground set. The function ξ maps superfamilies over A to families over A as follows. For any superfamily $\mathfrak{F}$:

$$\xi(\mathfrak{F}) \stackrel{\text{def}}{=} \{\cup Z | Z \in \mathfrak{F}\}$$

Syntactically speaking, ξ removes the innermost pairs of parentheses. For instance, suppose $A = \{a,b,c,d\}$ and $\mathfrak{F} = \{\{\{a\},\{b,c\}\},\{\{d\}\}\}$. Then $\xi(\mathfrak{F}) = \{\{a,b,c\},\{d\}\}$.

We now define the central function of the present study. It takes a partition of S, identifies the mincors of the corresponding quotient relation, merges the overlapping mincors, and uses ξ to map the result back to a partition of S.

Definition 5 (function α) $\alpha_R : \Pi(\mathrm{S}) \to \Pi(\mathrm{S})$*, shortly* α *when* R *is understood, is defined as follows for any* $\mathfrak{X} \in \Pi(\mathrm{S})$*:*

$$\alpha_R(\mathfrak{X}) \stackrel{def}{=} \xi(\mathtt{CC}(\mathtt{MF}(R/\mathfrak{X})))$$

Notably, α is *not monotone* in general as demonstrated by the following example. Let $\widetilde{\mathrm{S}} = \{A,B,C,D,E\}$ and let each attribute have precisely two values, say $A = \{a_1,a_2\}$ and so on. Let Q be the relation obtained from the complete relation over $\widetilde{\mathrm{S}}$ after deleting all tuples containing $a_1b_1c_1$, all tuples containing d_2e_2, and the tuples $\{a_2b_1c_1d_2e_1\},\{a_2b_2c_1d_2e_1\}$. In other words,

$$\begin{aligned} Q = \{&\{a_1b_1c_2d_1e_1\},\{a_1b_1c_2d_1e_2\},\{a_1b_1c_2d_2e_1\},\{a_1b_2c_1d_1e_1\},\{a_1b_2c_1d_1e_2\},\{a_1b_2c_1d_2e_1\}\\ &\{a_1b_2c_2d_1e_1\},\{a_1b_2c_2d_1e_2\},\{a_1b_2c_2d_2e_1\},\{a_2b_1c_1d_1e_1\},\{a_2b_1c_1d_1e_2\},\{a_2b_1c_2d_1e_1\}\\ &\{a_2b_1c_2d_1e_2\},\{a_2b_1c_2d_2e_1\},\{a_2b_2c_1d_1e_1\},\{a_2b_2c_1d_1e_2\},\{a_2b_2c_2d_1e_1\},\{a_2b_2c_2d_1e_2\},\{a_2b_2c_2d_2e_1\}\} \end{aligned}$$

Let us see which sets of attributes are self-correlated with respect to Q. The only two-element subset of $\widetilde{\mathrm{S}}$ that is self-correlated is $\{D,E\}$. Further, $\{A,B,C\}$ is self-correlated. It follows $\mathtt{MF}(Q) = \{\{A,B,C\},\{D,E\}\}$. Consider the following two partitions of $\widetilde{\mathrm{S}}$: $\mathfrak{X}_1 = \{\{A\},\{B\},\{C\},\{D\},\{E\}\}$ and $\mathfrak{X}_2 = \{\{A\},\{B,D\},\{C,E\}\}$. Obviously, $\mathfrak{X}_1 \sqsubseteq \mathfrak{X}_2$. It is clear that $\alpha(\mathfrak{X}_1) = \{\{A,B,C\},\{D,E\}\}$. Consider $\alpha(\mathfrak{X}_2)$. The set $\{\{B,D\},\{C,E\}\}$ is self-correlated because of the lack of $\{b_1,d_2\}$ and $\{c_1,e_2\}$ in any tuple, which in its turn is due to the fact that d_2 and e_2 do not occur in any tuple of R. The sets $\{\{A\},\{B,D\}\}$ and $\{\{A\},\{C,E\}\}$ are uncorrelated. It follows that $\alpha(\mathfrak{X}_2) = \{\{A\},\{B,C,D,E\}\}$, and thus $\alpha(\mathfrak{X}_1) \not\sqsubseteq \alpha(\mathfrak{X}_2)$.

However, we have the following property of α that shall later be exploited.

Proposition 4 α *is an inflationary function on* $(\Pi(\mathrm{S}),\sqsubseteq)$.

5.2 Independence and function α

The following central result establishes that the independent partitions are precisely the fixed points of α.

Theorem 1 $\forall \mathfrak{X} \in \Pi(\mathrm{S}) : \mathfrak{X} \in \mathrm{I}\Pi(\mathrm{S}) \leftrightarrow \alpha(\mathfrak{X}) = \mathfrak{X}$.

Proof: In one direction, assume $\mathfrak{X} \in \mathrm{I}\Pi(\mathrm{S})$. $R/\mathfrak{X}$ is complete by Proposition 1. By definition, that is $R/\mathfrak{X} = \mathbin{\mathbb{X}}_{Y\in\mathfrak{X}} Y$. By the definition of $\bowtie$, $(R/\mathfrak{X}) \upharpoonright \mathfrak{X} = \bowtie_{Y\in\mathfrak{X}} (R/\mathfrak{X}) \upharpoonright \{Y\}$. It follows that $\neg\mathrm{corr}(\mathfrak{X})$ by Lemma 1. So, $\mathsf{mincors}(R/\mathfrak{X}) = \emptyset$ and $\mathtt{MF}(R/\mathfrak{X}) = \mathsf{singletons}(R/\mathfrak{X})$ by Definition 4. Then $\mathtt{CC}(\mathtt{MF}(R/\mathfrak{X})) = \{\{A\} | A \in \mathfrak{X}\}$. Therefore, $\xi(\mathtt{CC}(\mathtt{MF}(R/\mathfrak{X}))) = \{A | A \in \mathfrak{X}\} = \mathfrak{X}$. But $\xi(\mathtt{CC}(\mathtt{MF}(R/\mathfrak{X})))$ is $\alpha(\mathfrak{X})$ by definition. Therefore, $\alpha(\mathfrak{X}) = \mathfrak{X}$.

In the other direction, assume $\alpha(\mathfrak{X}) = \mathfrak{X}$. That is, $\xi(\mathcal{CC}(\mathsf{MF}(R/\mathfrak{X}))) = \mathfrak{X}$, which in its turn implies $\mathcal{CC}(\mathsf{MF}(R/\mathfrak{X})) = \{\{A\} \mid A \in \mathfrak{X}\}$ because $\mathcal{CC}(\mathsf{MF}(R/\mathfrak{X}))$ is a superfamily such that every element from S is in precisely one element of precisely one element of it. The remainder of the proof mirrors the above one. □

Having in mind the observation on page 69 that $\mathrm{I}\Pi_R(\mathrm{S})$ coincides with $\uparrow\mathsf{foc}(R)$, we derive the following corollary of Theorem 1.

Corollary 2 *$\uparrow\mathsf{foc}(R)$ is closed with respect to α.*

The following lemma says that the mincors of a quotient relation respect the focus of the relation in the sense that for every mincor of $R/\mathfrak{X}$, the union of its elements is a subset of some element of the focus.

Lemma 7 $\forall \mathfrak{X} \in \downarrow\mathsf{foc}(R)\ \forall T \in \mathsf{mincors}(R/\mathfrak{X})\ \exists \mathrm{Y} \in \mathsf{foc}(R) : \cup T \subseteq \mathrm{Y}$.

Proof: Assume the contrary. That is, for some partition $\mathfrak{X}$ that refines the focus there is a mincor T of $R/\mathfrak{X}$ such that $\cup T$ has nonempty intersection with at least two subsets, call them Y_1 and Y_2, of the focus. Use Lemma 3 to conclude there is some $\mathrm{Z} \Subset T$ such that $|\mathrm{Z}| = |T|$ and $\cup\mathrm{Z} \in \mathsf{mincors}(R)$. Since $|\mathrm{Z}| = |T|$, it must be the case that $\cup\mathrm{Z}$ has nonempty intersection with both Y_1 and Y_2. But the focus is an independent partition. We derived that a mincor of R, namely $\cup\mathrm{Z}$, intersects two distinct elements of an independent partition. That contradicts Lemma 5 directly. □

We already established (see Proposition 4) that α is an inflationary function. The next lemma, however, establishes a certain restriction: the application of α on a dependent partition can yield another dependent partition or at most the focus, and never an independent partition "above" the focus.

Lemma 8 *$\downarrow\mathsf{foc}(R)$ is closed with respect to α.*

Proof: We prove that $\forall \mathfrak{X} \in \downarrow\mathsf{foc}(R) : \alpha(\mathfrak{X}) \sqsubseteq \mathsf{foc}(R)$. Recall that $\alpha(\mathfrak{X})$ is a partition of S and it abstracts $\mathfrak{X}$. Assume the claim is false. Then there is a partition $\mathfrak{X}$ such that $\mathfrak{X} \sqsubseteq \downarrow\mathsf{foc}(R)$ but $\alpha(\mathfrak{X}) \not\sqsubseteq \downarrow\mathsf{foc}(R)$. Then there is some $\mathrm{P} \in \alpha(\mathfrak{X})$ such that P has nonempty intersection with at least two elements, call them Y_1 and Y_2, of $\mathsf{foc}(R)$. However, P is $\xi(C)$ for some C that is a connected component—relative to the ground set $\mathfrak{X}$—of the mincor family of $R/\mathfrak{X}$. Consider C. It is the union of one or more mincors of $R/\mathfrak{X}$, those mincors being subsets of $\mathfrak{X}$.

Since $\mathfrak{X} \sqsubseteq \mathsf{foc}(R)$, no element of $\mathfrak{X}$ can intersect both Y_1 and Y_2. It follows that at least one mincor $\mathrm{M} \in C$ is such that $\cup\mathrm{M}$ intersects both Y_1 and Y_2. But that contradicts Lemma 7. □

The next and final central result allows us to compute the focus of R by an iterative application of α, starting with the partition into singletons.

Theorem 2 *For some m such that $1 \leq m \leq |\mathrm{S}|$, $\alpha^m(\bot) = \mathsf{foc}(R)$.*

Proof: Consider the sequence:

$$C = \bot,\ \alpha(\bot),\ \alpha^2(\bot),\ \ldots$$

It is a chain in the lattice $(\Pi(\mathrm{S}), \sqsubseteq)$, as $\alpha(\mathfrak{X})$ abstracts $\mathfrak{X}$ for all $\mathfrak{X}$ (see Proposition 4), therefore all those elements are comparable with respect to $\sqsubseteq$. C has only a finite number of distinct elements as the said lattice is finite.

First note that every element of C is in $\downarrow\mathsf{foc}(R)$. Indeed, assuming the opposite immediately contradicts Lemma 8.

Then note that for every $\mathfrak{X} \in \downarrow\mathsf{foc}(R) \setminus \{\mathsf{foc}(R)\}$, it is the case that $\alpha(\mathfrak{X}) \neq \mathfrak{X}$. Assuming the opposite implies $\mathfrak{X}$ is a fixed point of α, contradicting Corollary 2. Proposition 4 implies a stronger fact: for every

$\mathfrak{X} \in \downarrow\mathsf{foc}(R) \setminus \{\mathsf{foc}(R)\}$, it is the case that $\mathfrak{X} \sqsubset \alpha(\mathfrak{X})$. But $\downarrow\mathsf{foc}(R)$ is a finite lattice. It follows immediately that for some value m not greater than $|\mathrm{S}|$, $\alpha^m(\bot)$ equals the top of $\downarrow\mathsf{foc}(R)$, *viz.* $\mathsf{foc}(R)$. □

We thus obtain Kleene's iterative least fixed point approximation procedure [7], however for inflationary functions instead of monotone ones.

Corollary 3 *The following algorithm:*

$\mathfrak{X} \leftarrow \bot$
while $\mathfrak{X} \neq \alpha(\mathfrak{X})$
    $\mathfrak{X} \leftarrow \alpha(\mathfrak{X})$
return $\mathfrak{X}$

computes the least fixed point of α, i.e.*, the maximum independent partition of* S *with respect to* R. □

Here is a small example illustrating the work of that algorithm. Consider S and R' defined in (1) on page 64. $\bot$ is $\{\{A\},\{B\},\{C\},\{D\}\}$. Let us compute $\alpha(\bot)$, that is, $\xi(\mathtt{CC}(\mathtt{MF}(R'/\bot)))$. $R'/\bot$ is the same as $R'/\mathfrak{X}_2$ on page 64, namely:

$$R'/\bot = \{\{\{a_1\}\{b_1\}\{c_1\}\{d_1\}\},\{\{a_1\}\{b_1\}\{c_2\}\{d_2\}\},\{\{a_1\}\{b_2\}\{c_1\}\{d_2\}\},\\ \{\{a_2\}\{b_2\}\{c_1\}\{d_1\}\},\{\{a_2\}\{b_2\}\{c_2\}\{d_2\}\}\}$$

Let us compute $\mathtt{CC}(\mathtt{MF}(R'/\bot))$. Having in mind that $\mathtt{MF}(R') = \{\{A,B\},\{C,D\}\}$ as explained on page 65, conclude that $\mathtt{CC}(\mathtt{MF}(R'/\bot)) = \{\{\{A,B\}\},\{\{C,D\}\}\}$. Therefore, $\xi(\mathtt{CC}(\mathtt{MF}(R'/\bot))) = \{\{A,B\},\{C,D\}\}$. That differs from $\bot$ and the **while** loop is executed again. $R'/\alpha(\bot)$ is the same as $R'/\mathfrak{X}_1$ on page 64, namely:

$$R'/\alpha(\bot) = \{\{\{a_1b_1\}\{c_1d_1\}\},\{\{a_1b_1\}\{c_2d_2\}\},\{\{a_1b_2\}\{c_1d_2\}\},\\ \{\{a_2b_2\}\{c_1d_1\}\},\{\{a_2b_2\}\{c_2d_2\}\}\}$$

Let us compute $\mathtt{CC}(\mathtt{MF}(R'/\alpha(\bot)))$. To that end, note that $\alpha(\bot) = \{\{A,B\},\{C,D\}\}$ is self-correlated with respect to $R'/\{\{A,B\},\{C,D\}\}$ because of the lack of, for instance, both $\{a_1,b_2\}$ and $\{c_1,d_1\}$ in any tuple of $R'/\alpha(\bot)$. It follows that $\mathtt{CC}(\mathtt{MF}(R'/\alpha(\bot))) = \{\{\{A,B\},\{C,D\}\}\}$ and, therefore, $\alpha^2(\bot) = \xi(\mathtt{CC}(\mathtt{MF}(R'/\alpha(\bot)))) = \{\{A,B,C,D\}\}$. That differs from $\alpha(\bot)$ and the **while** loop is executed once more. At the end of that execution, it turns out that $\alpha^3(\bot)$ equals $\alpha^2(\bot)$ and the algorithm terminates, returning as the result $\{\{A,B,C,D\}\}$, the trivial partition.

6 Related Work

An algorithm that factorizes a given relation into prime factors is proposed in [10, algorithm PRIME FACTORIZATION]. It runs in time $O(mn\lg n)$ where m is the number of tuples and n is the number of attributes. Since mn is the input size, that time complexity is very close to the optimum. The theoretical foundation of PRIME FACTORIZATION is a theorem (see [10, Proposition 10]) that says a given relation S has a factor F iff, with respect to any attribute A and any value v of its domain, F is a factor of both Q and R where Q and R are relations such that $Q \cup R = S$ and Q consists precisely of the tuples in which the value of A is v. In other words, the approach of [10] to the problem of computing the prime factors is "horizontal splitting" of the given relation using the selection operation from relational algebra. The approach of this paper to that same problem is quite different. We utilise "vertical splitting", using the

projection operation of relational algebra. The theoretical foundation of our approach is based on the concept of self-correlation of a subset of the attributes; that concept has no analogue in [10].

An excellent exposition of the benefits of the factorisation of relational data is [11]. The factorised representation both saves space, where the gain can potentially be as good as exponential, and time, speeding up the processing of information whose un-factorised representation is too big. [1] proposes a way of decomposing relational data that is incomplete and [13] proposes factorisation of relational data that facilitates machine learning.

Clusterisation of multidimensional data into non-intersecting classes called clusters is an important, hard and computationally demanding problem. [5] investigates clustering in high-dimensional data by detection of orthogonality in the latter. [8] proposes so called community discovering, which is a sort of clusterisation, in media social networks by utilising factorisation of a relational hypergraph.

The foundation of this paper is the work of Gurov *et al.* [6] that investigates relational factorisation of a restricted class of relations called there simple families. [6] introduces the concept of correlation between the attributes and proposes a fast and practical algorithm that computes the optimum factorisation of a simple family by using a subroutine for correlation. The fundamental approach of this paper is an extension of that, however now correlation is considerately more involved, being not a binary relation between attributes but a relation of arbitrary arity (this is the only place where "relation" means relation in the Set Theory sense, that is, a set of ordered tuples).

7 Conclusion

This paper illustrates the utility of fixed points to formally express maximum independence in relations by means of minimum correlated sets of attributes. By using minimum correlated sets, we define an inflationary transformer over a finite lattice and show the maximum independent partition is the least fixed point of this transformer. Then we prove the downward closure of that least fixed point is closed under the transformer. Hence, the least fixed point can be computed by applying the transformer iteratively from the bottom element of the lattice until stabilization. This iterative construction is the same as Kleene's construction, but does not rely on monotonicity of the transformer to guarantee that it computes the least fixed point.

A topic for future work is to introduce a quantitative measure for the degree of independence between sets of attributes and investigate approximate relational factorisation.

Acknowledgement We are indebted to Zoltán Ésik for pointing out to us that the CPO Fixpoint Theorem III of [3, pp. 188] about the existence of least fixed points of inflationary functions (called there "increasing functions") in CPOs does in fact not hold, and to Valentin Goranko for directing us to the work by Olteanu et al. Finally, we thank the reviewers of this paper for the thorough assessments and the valuable suggestions that allowed us to improve the quality of the presentation.

References

[1] Lyublena Antova, Christoph Koch & Dan Olteanu (2009): *$10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information*. *VLDB J.* 18(5), pp. 1021–1040, doi:10.1007/s00778-009-0149-y.

[2] Garrett Birkhoff (1967): *Lattice Theory*, 3rd edition. American Mathematical Society, Providence.

[3] Brian A. Davey & Hilary A. Priestley (2002): *Introduction to Lattices and Order*. Cambridge mathematical text books, Cambridge University Press, doi:10.1017/CBO9780511809088.

[4] Erich Grädel & Jouko A. Väänänen (2013): *Dependence and Independence*. *Studia Logica* 101(2), pp. 399–410, doi:10.1007/s11225-013-9479-2.

[5] Stephan Günnemann, Emmanuel Müller, Ines Färber & Thomas Seidl (2009): *Detection of Orthogonal Concepts in Subspaces of High Dimensional Data*. In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, ACM, New York, NY, USA, pp. 1317–1326, doi:10.1145/1645953.1646120.

[6] Dilian Gurov, Bjarte M. Østvold & Ina Schaefer (2011): *A Hierarchical Variability Model for Software Product Lines*. In: *Leveraging Applications of Formal Methods, Verification, and Validation - International Workshops, SARS 2011 and MLSC 2011, Held Under the Auspices of ISoLA 2011 in Vienna, Austria, October 17-18, 2011. Revised Selected Papers*, pp. 181–199, doi:10.1007/978-3-642-34781-8_15.

[7] Jean-Louis Lassez, V. L. Nguyen & Liz Sonenberg (1982): *Fixed Point Theorems and Semantics: A Folk Tale*. *Information Processing Letters* 14(3), pp. 112–116, doi:10.1016/0020-0190(82)90065-5.

[8] Yu-Ru Lin, Jimeng Sun, Paul Castro, Ravi Konuru, Hari Sundaram & Aisling Kelliher (2009): *MetaFac: Community Discovery via Relational Hypergraph Factorization*. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, ACM, New York, NY, USA, pp. 527–536, doi:10.1145/1557019.1557080.

[9] David Maier (1983): *The Theory of Relational Databases*. Computer Science Press. Accessible online at `http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html`.

[10] Dan Olteanu, Christoph Koch & Lyublena Antova (2008): *World-set decompositions: Expressiveness and efficient algorithms*. *Theoretical Computer Science* 403(2-3), pp. 265–284, doi:10.1016/j.tcs.2008.05.004.

[11] Dan Olteanu & Jakub Závodný (2015): *Size Bounds for Factorised Representations of Query Results*. *ACM Trans. Database Syst.* 40(1), p. 2, doi:10.1145/2656335.

[12] Klaus Pohl, Günter Böckle & Frank van der Linden (2005): *Software Product Line Engineering - Foundations, Principles, and Techniques*. Springer, doi:10.1007/3-540-28901-1.

[13] Steffen Rendle (2013): *Scaling Factorization Machines to Relational Data*. *PVLDB* 6(5), pp. 337–348. Available at `http://www.vldb.org/pvldb/vol6/p337-rendle.pdf`.

[14] S. Roman (2008): *Lattices and Ordered Sets*. Springer. Available at `https://books.google.com/books?id=NZN8aum26LgC`.

[15] Ina Schaefer, Dilian Gurov & Siavash Soleimanifard (2010): *Compositional Algorithmic Verification of Software Product Lines*. In: *Formal Methods for Components and Objects - 9th International Symposium, FMCO 2010, Graz, Austria, November 29 - December 1, 2010. Revised Papers*, pp. 184–203, doi:10.1007/978-3-642-25271-6_10.

[16] Alfred Tarski (1955): *A Lattice-Theoretical Fixpoint Theorem and Its Applications*. *Pacific journal of Mathematics* 5(2), pp. 285–309, doi:10.2140/pjm.1955.5.285.

[17] Bruce W. Weide, Wayne D. Heym & Joseph E. Hollingsworth (1995): *Reverse Engineering of Legacy Code Exposed*. In: *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, ACM, New York, NY, USA, pp. 327–331, doi:10.1145/225014.225045.

Iteration Algebras for UnQL Graphs and Completeness for Bisimulation

Makoto Hamana
Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp

This paper shows an application of Bloom and Ésik's iteration algebras to model graph data in a graph database query language. About twenty years ago, Buneman et al. developed a graph database query language UnQL on the top of a functional meta-language UnCAL for describing and manipulating graphs. Recently, the functional programming community has shown renewed interest in UnCAL, because it provides an efficient graph transformation language which is useful for various applications, such as bidirectional computation. However, no mathematical semantics of UnQL/UnCAL graphs has been developed. In this paper, we give an equational axiomatisation and algebraic semantics of UnCAL graphs. The main result of this paper is to prove that completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. Another benefit of algebraic semantics is a clean characterisation of structural recursion on graphs using free iteration algebra.

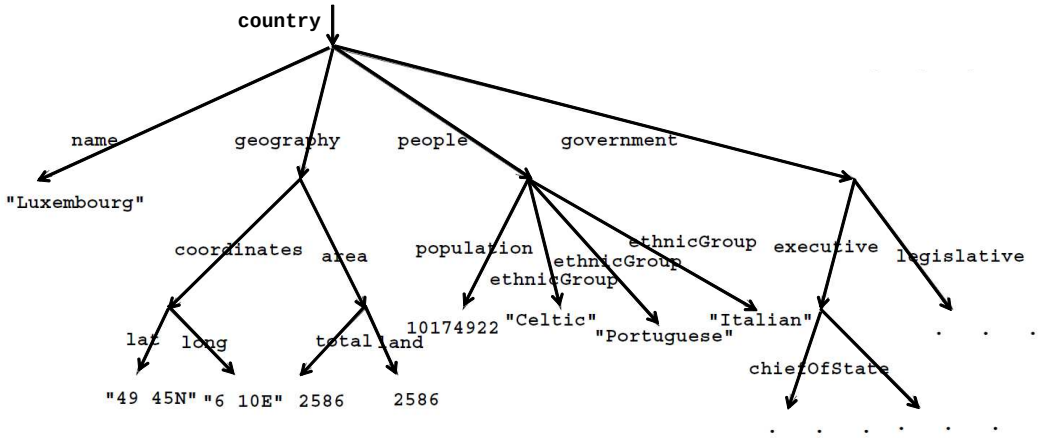
1 Introduction

Graph database is used as a back-end of various web and net services, and therefore it is one of the important software systems in the Internet society. About twenty years ago, Buneman et al. [6, 7, 8] developed a graph database query language UnQL (Unstructured data Query Language) on top of a functional meta-language **UnCAL** (Unstructured Calculus) for describing and manipulating graph data. The term "unstructured" is used to refer to unstructured or semi-structured data, i.e., data having no assumed format in a database (in contrast to relational database). Recently, the functional programming community found a new application area of UnCAL in so-called bidirectional transformations on graph data, because it provides an efficient graph transformation language. The theory and practice of UnCAL have been extended and refined in various directions (e.g. [18, 19, 17, 1]), which has increased the importance of UnCAL.

In this paper, we give a more conceptual understanding of UnCAL using semantics of type theory and fixed points. We give an equational axiomatisation and algebraic semantics of UnCAL graphs. The main result of this paper is to prove completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. Another benefit of algebraic semantics is a clean characterisation of the computation mechanism of UnCAL called "structural recursion on graphs" using free iteration algebra.

UnCAL Overview. We begin by introducing UnCAL. UnCAL deals with graphs in a graph database. Hence, it is better to start with viewing how concrete semi-structured data is processed in UnCAL. Consider the semi-structured data `sd` below which is taken from [8].

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 75–89, doi:10.4204/EPTCS.191.8



It contains information about country, e.g. geography, people, government, etc. It is depicted as a tree above, in which edges and leaves are labelled. Using UnCAL's term language for describing graphs (and trees), this is defined by `sd` shown at right. Then we can define functions in UnCAL to process data. For example, a function that retrieves all ethnic groups in the graph can be defined simply by

```
sd ◃ country:{name:"Luxembourg",
 geography:{coordinates:{long:"49 45N", lat:"6 10E"},
            area:{total:2586, land:2586}},
 people:{population:425017,
         ethnicGroup:"Celtic",
         ethnicGroup:"Portuguese",
         ethnicGroup:"Italian"},
 government:{executive:{chiefOfState:{name:"Jean",..}}}}
```

```
sfun f1(L:T) = if L = ethnicGroup then (result:T) else f1(T)
```

The keyword `sfun` denotes a function definition by *structural recursion on graphs*, which is the computational mechanism of UnCAL. Executing it, we can certainly extract:

```
f1(sd) ⇝ {result:"Celtic", result:"Portuguese", result:"Italian"}
```

The notation $\{\cdots:\cdots,\cdots\}$ is a part of the UnCAL's term language for representing graphs. It consists of markers x, labelled edges $\ell{:}t$, vertical compositions $s \diamond t$, horizontal compositions $\langle s, t\rangle$, other horizontal compositions $s \cup t$ merging roots, forming cycles cycle(t), constants {},(), and definitions ($x \triangleleft t$). These term constructions have underlying graph theoretic meaning shown at th right. Namely, these are officially defined as operations on the ordinary representations of graphs: (vertices set, edges set, leaves, roots)-tuples $(V, E, \{y_1, \ldots y_m\}, \{x_1, \ldots, x_n\})$, but we do not use the graph theoretic definitions of these operations in this paper.

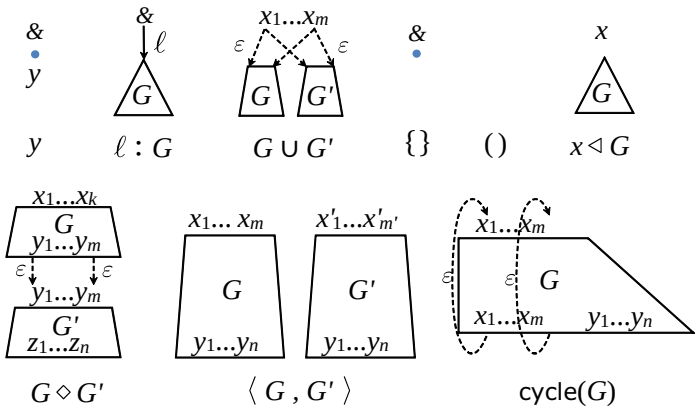


Figure 1: Graph theoretic definitions of constructors [8]
Slightly changed notation. Correspondence between the original and this paper's: $\&y = y$, $@ = \diamond$, $\oplus = \langle -, - \rangle$, $(- := -) = - \triangleleft -$.

UnCAL deals with graphs *modulo bisimulation* (i.e. not only modulo graph isomorphism). An UnCAL graph is directed and have (possibly multiple) root(s) written & (or multiple $x_1 \cdots x_n$) and leaves (written $y_1 \cdots y_m$), and with the roots and leaves drawn pictorially at the top and bottom, respectively. The symbols $x, y_1, y_2, \&$ in the figures and terms are called markers, which are the names of nodes in a graph and are used for references for cycles. Also, they are used as port names to connect two graphs. A dotted line labelled ε is called an ε-edge, which is a "virtual" edge connecting two nodes directly. This is achieved by identifying graphs by *extended bisimulation*, which ignores ε-edges suitably in UnCAL. The UnCAL graph G shown at the left is an example. This is extended bisimilar to a graph that reduces all ε-edges. Using UnCAL's language, G is represented as the following term $\mathbf{t}_G$

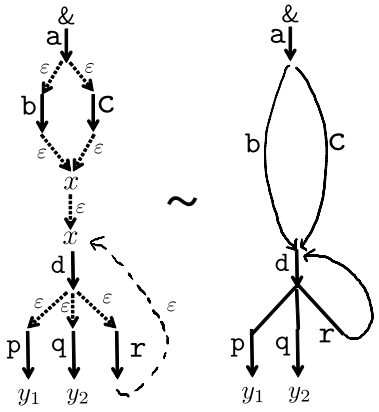


Figure 2: Graph G and bisimilar one

$$\mathbf{t}_G \;=\; \mathtt{a}\!:\!(\{\mathtt{b}\!:\!x\} \cup \{\mathtt{c}\!:\!x\}) \diamond \mathsf{cycle}(x \triangleleft \mathtt{d}\!:\!(\{\mathtt{p}\!:\!y_1\} \cup \{\mathtt{q}\!:\!y_2\} \cup \{\mathtt{r}\!:\!x\})\,).$$

UnCAL's structural recursive function works also on cycle. For example, define another function

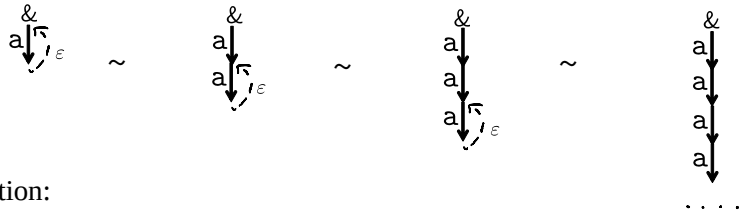
```
sfun f2(L:T) = a:f2(T)
```

that replaces every edge with a. As expected,

$$\mathtt{f2}(\mathbf{t}_G) \;\rightsquigarrow\; \mathtt{a}\!:\!(\{\mathtt{a}\!:\!x\} \cup \{\mathtt{a}\!:\!x\}) \diamond \mathsf{cycle}(x \triangleleft \mathtt{a}\!:\!(\{\mathtt{a}\!:\!y_1\} \cup \{\mathtt{a}\!:\!y_2\} \cup \{\mathtt{a}\!:\!x\})\,)$$

where all labels are changed to a.

Another characteristic role of bisimulation is that it identifies expansion of cycles. For example, a term $\mathsf{cycle}(\& \triangleleft \mathtt{a}\!:\!\&)$ corresponds to the graph shown below at the leftmost. It is bisimilar to the right ones, especially the infinitely expanded graph shown at the rightmost, which has no cycle.



These are in term notation:

$$\mathsf{cycle}(\& \triangleleft \mathtt{a}\!:\!\&) \;\sim\; \mathtt{a}\!:\!\mathsf{cycle}(\& \triangleleft \mathtt{a}\!:\!\&) \;\sim\; \mathtt{a}\!:\!\mathtt{a}\!:\!\mathsf{cycle}(\& \triangleleft \mathtt{a}\!:\!\&)$$

Problems. There have been no algebraic laws that establish the above expansion of cycle. Namely, these are merely bisimilar, and not a consequence of any algebraic law. But obviously, we expect that it should be a consequence of the algebraic law of *fixed point property* of cycle.

In the original and subsequent formulation of UnCAL [8, 17, 18, 1], there are complications of this kind. The relationship between terms and graphs in UnCAL is not a one-to-one correspondence. No term notation exits for ε-edges and infinite graphs (generated by the cycle construct), thus the rightmost infinite graphs of the above expansion cannot be expressed in syntax. But such an infinite graph is allowed as a possible graph in the original formulation of UnCAL. Consequently, instead of terms, one must use graphs and graph theoretic reasoning with care of bisimulation to reason about UnCAL. Therefore, a property in UnCAL could not be established only using induction on terms. That fact sometime makes some proofs about UnCAL quite complicated.

Because UnCAL graphs are identified by bisimulation, it is necessary to use a procedure or algorithm to check the bisimilarity as in the cycle example above. Listing some typical valid equations for the bisimulation can be a shortcut [8, 19], but it was only sound and *not complete* for bisimulation.

Hence, we give an algebraic and type-theoretic formulation of UnCAL by giving equational axioms of UnCAL graphs. In this paper, we prove completeness of our proposed axioms using iteration algebra [4]. Thus we have a *complete* syntactic axiomatisations of the equality on UnQL/UnCAL graphs, as a set of axioms capturing the original bisimulation, without touching graphs, ε-edges, and the notion of bisimulation explicitly. We prove it by connecting it with the algebraic axiomatisations of bisimulation [3, 12].

How to model UnCAL and structural recursion. The first idea to understand UnCAL is to interpret it as a categorical structure. We can regard edges as *morphisms* (of the opposite directions), the vertical composition $\diamond$ as the *composition of arrows*, and cycle as a *fixpoint operator* in a suitable category. Thus the target categorical structure should have a notion of fixpoint, which has been studied in iteration theories of Bloom and Ésik [3]. In particular, iteration categories [10] are suitable, which are traced cartesian categories [20] (monoidal version is used in Hasegawa's modelling of cyclic sharing theories [16, 15]) additionally satisfying the commutative identities axiom [3] (see also [25] Section 2 for a useful overview around this).

We also need to model UnCAL's computational mechanism: "structural recursion on graphs". The general form of the definition of structural recursive function is

$$\texttt{sfun}\ F(\ell : t) = e \qquad (\bigstar)$$

where e can involve $F(t)$. The graph algorithm in [8] provide a transformation of graphs that produces some computed graphs using the definition ($\bigstar$). It becomes a function F satisfying the equations ([8] Prop. 3):

$$\begin{array}{lll} F(\,y_i\,) = y_i & F(\,(x \triangleleft t)\,) = (x \triangleleft F(t)) & F(\quad \ell\!:\!t \quad) = e \\ F(\,()\,) = () & F(\quad s \cup t \quad) = F(s) \cup F(t) & F(\quad s \diamond t \quad) = F(s) \diamond F(t) \quad \cdots(\bowtie) \\ F(\,\{\}\,) = \{\} & F(\ \langle s, t \rangle\ ) = \langle F(s), F(t) \rangle & F(\ \mathsf{cycle}(t)\ ) = \mathsf{cycle}(F(t)) \quad \cdots(\bowtie) \end{array} \qquad (1)$$

when e does not depend[1] on t. This is understandable naturally as the example `f2` recurses structurally the term $\mathbf{t}_G$. Combining the above categorical viewpoint, F can be understood as a functor that preserves cycle and products (thus a traced cartesian functor). A categorical semantics of UnCAL can be given along this idea, which will be reported elsewhere. This idea works for simple cases of structural recursion such as `f2`.

However, there is a critical mismatch between the above categorical view and UnCAL's structural recursion of more involved cases. Buneman et al. mentioned a condition that the above nine equations hold only when *e does not* depend on t in ($\bigstar$). Two equations marked ($\bowtie$) do not hold in general if *e does* depend on t (other seven equations do hold). Crucially, `f1` is already this case, where `T` appears as not of the form `f1(T)`. The following another example shows why ($\bowtie$) do not hold: the structural recursive function `aa?` tests whether the argument contains "`a:a:`".

```
sfun a?(L:T)  = if L=a then true:{} else {}
sfun aa?(L:T) = if L=a then a?(T) else aa?(T)
```

The definition of `aa?` *does* depend on `T` at the "then"-clause. Then we have the inequalities:

[1] Here "e depends on t" means that e contains t other than the form $F(t)$.

```
aa?( (a:&)⋄(a:{})) = aa?( a:a:{} ) = true:{} ≠ {} = {}⋄{} = aa?(a:&) ⋄ aa?(a:{})
aa?( cycle(a:&) ) = aa?( a:a:cycle(a:&) ) = true:{} ≠ {} = cycle({}) = cycle(aa?(a:&))
```

This means that F does not preserve cycle in general, and even *is not functorial*, thus the categorical view seems not helpful to understand this pattern of recursion.

In this paper, we consider *algebraic semantics* of UnCAL using the notion of iteration Σ-algebras [4, 12] in §3. It solve the problem mentioned above, i.e. we derive the structural recursion even when the case that e depends on t within the algebraic semantics.

Organisation. This paper is organised as follows. We first give a framework of equational theory for UnCAL graphs by reformulating UnCAL graph data in a type theoretic manner in Section 2. We then give algebraic semantics of UnCAL using iteration Σ-algebras in Section 3. We prove completeness of our axioms for UnCAL graphs for bisimulation in Section 3.3. We further derive structural recursion on UnCAL graphs in Section 3.5. Finally, in Section 3.6. we show several examples how structural recursive functions on graphs are modeled.

2 UnCAL and its Equational Theory

We give a framework of equational theory for UnCAL graphs. We reformulate UnCAL graph data in a type theoretic manner. We do not employ the graph theoretic and operational concepts (such as ε-edges, bisimulation, and the graph theoretic definitions in Fig. 1). Instead, we give an algebraic axiomatisation of UnCAL graphs following the tradition of categorical type theory [9]. The syntax in this paper is slightly modified from the original presentation [8] to reflect the categorical idea, which may be more readable for the reader familiar with categorical type theory.

2.1 Syntax

Markers and contexts. We assume an infinite set of symbols called *markers*, denoted by typically $x, y, z, \ldots$. One can understand markers as variables in a type theory. The marker denoted by & is called the default marker, which is just a default choice of a marker having no special property. Let L be a set of *labels*. A *label* ℓ is a symbol (e.g. $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ in Fig. 2). A *context*, denoted by $\langle\langle x_1, x_2, \ldots\rangle\rangle$, is a sequence of pairwise distinct markers . We typically use $X, Y, Z, \ldots$ for contexts. We use $\langle\langle\rangle\rangle$ for the empty contexts, X, Y for the concatenation, and $|X|$ for its length. We may use the vector notation $\vec{x}$ for sequence $x_1, \ldots, x_n$. The outermost bracket $\langle\langle\ \rangle\rangle$ of a context may be omitted. We may use the abbreviations for the empty context $0 = \langle\langle\rangle\rangle$. Note that the concatenation may need suitable renaming to satisfy pairwise distinctness of markers.

Raw terms.

$$t \ ::= \ y_Y \ \mid \ \ell{:}t \ \mid \ s \diamond t \ \mid \ \langle s, t\rangle \ \mid \ \mathsf{cycle}^X(t) \ \mid \ \{\}_Y \ \mid \ ()_Y \ \mid \ \curlywedge \ \mid \ (x \triangleleft t)$$

We assume several conventions to simplify the presentation of theory. We often omit subscripts or superscripts such as Y when they are unimportant or inferable. We identify $\langle\langle s, t\rangle, u\rangle$ with $\langle s, \langle t, u\rangle\rangle$; thus we will freely omit parentheses as $\langle t_1, \ldots, t_n\rangle$. A constant $\curlywedge$ express a branch in a tree, and we call the symbol $\curlywedge$ a *man*, because it is similar to the shape of a kanji or Chinese character meaning a man, which is originated from the figure of a man having two legs (and the top is a head).

$$\text{(Nil)}\ \frac{}{Y \vdash \{\}_Y\ :\&} \qquad \text{(Emp)}\ \frac{}{Y \vdash ()_Y\ :\langle\langle\rangle\rangle} \qquad \text{(Man)}\ \frac{}{y_1, y_2 \vdash \curlywedge_{\langle\langle y_1, y_2\rangle\rangle}\ :\&}$$

$$\text{(Com)}\ \frac{Y \vdash s\ :Z \qquad X \vdash t\ :Y}{X \vdash s \diamond t\ :Z} \qquad \text{(Label)}\ \frac{\ell \in L \quad Y \vdash t\ :\&}{Y \vdash \ell\!:\! t\ :\&} \qquad \text{(Mark)}\ \frac{Y = \langle\langle y_1, \ldots, y_n\rangle\rangle}{Y \vdash y_{iY}\ :\&}$$

$$\text{(Pair)}\ \frac{Y \vdash s\ :X_1 \qquad Y \vdash t\ :X_2}{Y \vdash \langle s, t\rangle\ :X_1, X_2} \qquad \text{(Cyc)}\ \frac{Y, X \vdash t\ :X}{Y \vdash \mathsf{cycle}^X(t)\ :X} \qquad \text{(Def)}\ \frac{Y \vdash t\ :\&}{Y \vdash (x \triangleleft t)\ :x}$$

Figure 3: Typing rules

Abbreviations. We use the following abbreviations.

$$\begin{array}{llllll}
\{s\} \cup \{t\} & \triangleq & \curlywedge \diamond \langle s, t\rangle & s \times t & \triangleq & \langle s \diamond \pi_1, t \diamond \pi_2\rangle \\
\pi_1 & \triangleq & x_{\langle\langle x,y\rangle\rangle} & \mathsf{id}_{\langle\langle x\rangle\rangle} & \triangleq & x_{\langle\langle x\rangle\rangle} \\
\pi_2 & \triangleq & y_{\langle\langle x,y\rangle\rangle} & \mathsf{id}_{\langle\langle x_1,\ldots,x_n\rangle\rangle} & \triangleq & x_{1\,\langle\langle x_1\rangle\rangle} \times \cdots \times x_{n\,\langle\langle x_n\rangle\rangle}
\end{array}$$

$$\Delta_X \triangleq \langle \mathsf{id}_X, \mathsf{id}_X\rangle \qquad \mathsf{c} \triangleq \langle \pi_2, \pi_1\rangle$$

Inheriting the convention of $\langle -, -\rangle$, we also identify $(s \times t) \times u$ with $s \times (t \times u)$, thus we omit parentheses as $t_1 \times \ldots \times t_n$.

2.2 Typed syntax

For contexts X, Y, we inductively define a judgment relation $Y \vdash t\ :X$ of terms by the typing rules in Fig. 3. We call a marker *free* in t when it occurs in t other than the left hand-side of a definition ($x \triangleleft s$). In a judgment, free markers in t are always taken from Y. Thus Y is a variable context (which we call the *source context*) in ordinary type theory, and X is the roots (which we call the *target context* or *type*). For example, the term $\mathbf{t_G}$ in §1 is well-typed $y_1, y_2 \vdash \mathbf{t_G}\ :\&$, which corresponds a graph in Fig. 2, where the marker $\&$is the name of the root. When t is well-typed by the typing rules, we call t a (well-typed UnCAL) term. We identify t of type $\&$ with $(\& \triangleleft t)$.

Definition 2.1 (Substitution) Let $Y = \langle\langle y_1 \cdots, y_k\rangle\rangle$, W be contexts such that $|Y| \leq |W|$ and Y can be embedded into W in an order-preserving manner, and Y' is the subsequence of W deleting all of Y (NB. $|W| = |Y| + |Y'|$, Y' is possibly empty). Suppose $W \vdash t\ :X$, $Z \vdash s_i\ :\langle\langle y_i\rangle\rangle$ $(1 \leq i \leq k)$. Then a substitution $Z, Y' \vdash t\,[\vec{y} \mapsto \vec{s}]\ :X$ is inductively defined as follows.

$$\begin{array}{lll}
y_i\,[\vec{y} \mapsto \vec{s}] \triangleq s_i & & (t_1 \diamond t_2)\,[\vec{y} \mapsto \vec{s}] \triangleq t_1 \diamond (t_2\,[\vec{y} \mapsto \vec{s}]) \\
x\,[\vec{y} \mapsto \vec{s}] \triangleq x\ (\text{if } x \text{ in } Y') & & \langle t_1, t_2\rangle\,[\vec{y} \mapsto \vec{s}] \triangleq \langle (t_1\,[\vec{y} \mapsto \vec{s}]), (t_2\,[\vec{y} \mapsto \vec{s}])\rangle \\
\{\}_Y\,[\vec{y} \mapsto \vec{s}] \triangleq \{\}_{Z+Y'} & & \mathsf{cycle}(t)\,[\vec{y} \mapsto \vec{s}] \triangleq \mathsf{cycle}(t\,[\vec{y} \mapsto \vec{s}]) \\
()_Y\,[\vec{y} \mapsto \vec{s}] \triangleq ()_{Z+Y'} & & (x \triangleleft t)\,[\vec{y} \mapsto \vec{s}] \triangleq (x \triangleleft t\,[\vec{y} \mapsto \vec{s}]) \\
(\ell\!:\! t)\,[\vec{y} \mapsto \vec{s}] \triangleq \ell\!:\!(t\,[\vec{y} \mapsto \vec{s}]) & & \\
\curlywedge_{\langle\langle y_1,y_2\rangle\rangle}\,[y_1 \mapsto s_1, y_2 \mapsto s_2] \triangleq \curlywedge_{\langle\langle y_1,y_2\rangle\rangle} \diamond (s_1, s_2) & &
\end{array}$$

Note that $t\,[\vec{y} \mapsto \vec{s}]$ denotes a meta-level substitution operation, not an explicit substitution.

2.3 Equational theory

For terms $Y \vdash s\ :X$ and $Y \vdash t\ :X$, an *(UnCAL) equation* is of the form $Y \vdash s = t\ :X$. Hereafter, for simplicity, we often omit the source X and target Y contexts, and simply write $s = t$ for an equation, but even such an abbreviated form, we assume that it has implicitly suitable source and target contexts and is of the above judgemental form.

Composition

(sub1) $t \diamond (y \triangleleft s) = t\,[y \mapsto s]$ for $y \vdash t : X$

Parameterised fixpoint

(fix) $\mathsf{cycle}(t) = t \diamond \langle \mathsf{id}_Y, \mathsf{cycle}(t)\rangle$

(Bekič) $\mathsf{cycle}(\langle t, s\rangle) = \langle \pi_2, \mathsf{cycle}(s)\rangle \diamond \langle \mathsf{id}_Y, \mathsf{cycle}(t \diamond \langle \mathsf{id}_{Y,X}, \mathsf{cycle}(s)\rangle)\rangle$

(nat$_Y$) $\mathsf{cycle}(t) \diamond s = \mathsf{cycle}(t \diamond (s \times \mathsf{id}_X))$

(nat$_X$) $\mathsf{cycle}(s \diamond t) = s \diamond \mathsf{cycle}(t \diamond (\mathsf{id}_Y \times s))$

(CI) $\mathsf{cycle}(\langle t \diamond (\mathsf{id}_X \times \rho_1), \ldots, t \diamond (\mathsf{id}_X \times \rho_m)\rangle) = \Delta_m \diamond \mathsf{cycle}(t \diamond (\mathsf{id}_X \times \Delta_m))$

Deleting trivial cycle

(c2) $\mathsf{cycle}(\curlywedge) = \mathsf{id}$

Commutative monoid

(unitL$\curlywedge$) $\curlywedge \diamond (\{\}_0 \times \mathsf{id}) = \mathsf{id}$

(assoc$\curlywedge$) $\curlywedge \diamond (\mathsf{id} \times \curlywedge) = \curlywedge \diamond (\curlywedge \times \mathsf{id})$

(com$\curlywedge$) $\curlywedge \diamond \mathsf{c} = \curlywedge$

Degenerated bialgebra

(compa) $\Delta \diamond \curlywedge = (\curlywedge \times \curlywedge) \diamond (\mathsf{id} \times \mathsf{c} \times \mathsf{id}) \diamond (\Delta \times \Delta)$

(degen) $\curlywedge \diamond \Delta = \mathsf{id}$

Figure 4: Axioms AxGr for UnCAL graphs

Fig. 4 shows *our proposed axioms* AxGr to characterise UnCAL graphs. These axioms are chosen to soundly and completely represent the original bisimulation of graphs by the equality of this logic. Actually, it is sound: for every axiom $s = t$, s and t are bisimilar. But completeness is not clear only from the axioms. We will show it in §3.

The axiom (sub1) is similar to the β-reduction in the λ-calculus, which induces the axioms for cartesian product (cf. the **derived theory** below). The cartesian structure provides a canonical commutative comonoid with comultiplication Δ.

Two terms are paired with a common root by $\{s\} \cup \{t\} = \curlywedge \diamond (s, t)$. The commutative monoid axioms states that this pairing $\{-\} \cup \{-\}$ can be parentheses free in nested case. The degenerate bialgebra axioms state the compatibility between the commutative monoid and comonoid structures. The degenerated bialgebra is suitable to model directed acyclic graphs (cf. [14] §4.5), where it is stated within a PROP [21]. The monoid multiplication $\curlywedge$ expresses a branch in a tree, while the comultiplication Δ expresses a sharing. Commutativity expresses that there is no order between the branches of a node, cf. (commu$\cup$) in the **derived theory** below, and degeneration expresses that the branches of a node form a set (not a sequence), cf. (degen').

Parameterised fixpoint axioms axiomatise a fixpoint operator. They (minus (CI)) are known as the axioms for Conway operators of Bloom and Ésik [3], which ensures that all equalities that holds in cpo semantics do hold. It is also arisen in work independently of Hyland and Hasegawa [15], who established a connection with the notion of traced cartesian categories [20]. There are equalities that Conway operators do not satisfy, e.g. $\mathsf{cycle}(t) = \mathsf{cycle}(t \diamond t)$ does not hold only by the Conway axioms. The axiom (CI) fills this gap, which corresponds to the commutative identities of Bloom and Ésik [3]. This form is taken from [25] and adopted to the UnCAL setting, where $\Delta_m \triangleq \langle \mathsf{id}_\&, \cdots, \mathsf{id}_\&\rangle$, $Y = \langle\langle y_1, \ldots, y_m\rangle\rangle$, $\& \vdash \Delta_m : Y$, $X + Y \vdash t : \&$, $Y \vdash \rho_i : Y$ such that $\rho_i = \langle q_{i1}, \ldots, q_{im}\rangle$ where each q_{ij} is one of $Y \vdash \pi_i : \&$ for $i = 1, \ldots, m$. The axiom (c2) (and derived (c1) below) have been taken as necessary ones for completeness for bisimulation used in several axiomatisations, e.g. [23, 5, 12].

The equational logic EL-UnCAL for UnCAL is a logic to deduce formally proved equations, called *(UnCAL) theorems*. The equational logic is almost the same as ordinary one for algebraic terms. The inference rule of the logic consists of reflexivity, symmetricity, transitivity, congruence rules for all constructors, with the following axiom and the substitution rules.

$$\text{(Ax)}\ \frac{(Y \vdash s = t : X) \in E}{Y \vdash s = t : X} \qquad \text{(Sub)}\ \frac{W \vdash t = t' : X \qquad Z \vdash s_i = s_i' : y_i\ (1 \le i \le k)}{Z + Y' \vdash t\,[\vec{y} \mapsto \vec{s}] = t'\,[\vec{y} \mapsto \vec{s'}] : X}$$

The set of all theorems deduced from the axioms AxGr is called a *(UnCAL) theory*.

Derived theory. The following are formally derivable from the axioms, thus are theorems.

(tmnl)	$t = ()_Y$ for all $Y \vdash t : \langle\langle\rangle\rangle$			(dpair)	$\langle t_1, t_2\rangle \diamond s$	$=$	$\langle t_1 \diamond s, t_2 \diamond s\rangle$
(fst)	$\pi_1 \diamond \langle s, t\rangle$	$=$	s	(fsi)	$\langle \pi_1, \pi_2\rangle$	$=$	id
(snd)	$\pi_2 \diamond \langle s, t\rangle$	$=$	t	(SP)	$\langle \pi_1 \diamond t, \pi_2 \diamond t\rangle$	$=$	t

(bmul)	$()_\& \times ()_\&$	$=$	$()_\& \diamond \curlywedge$	(bcomul)	$\Delta \diamond \{\}_0$	$=$	$(\{\}_0 \times \{\}_0)$
(unitR$\curlywedge$)	$\curlywedge \diamond (\mathsf{id} \times \{\}_0)$	$=$	id	(bunit)	$()_\& \diamond \{\}_0$	$=$	id
(c1)	$\mathsf{cycle}(\mathsf{id})$	$=$	$\{\}_0$	(comm$\cup$)	$\{s\} \cup \{t\}$	$=$	$\{t\} \cup \{s\}$
(unR$\diamond$)	$t \diamond \mathsf{id}$	$=$	t	(unit$\cup$)	$\{\{\}\} \cup \{t\}$	$=$	$t = \{t\} \cup \{\{\}\}$
(unL$\diamond$)	$\mathsf{id} \diamond t$	$=$	t	(assoc$\cup$)	$\{\{s\} \cup \{t\}\} \cup \{u\}$	$=$	$\{s\} \cup \{\{t\} \cup \{u\}\}$
(assoc$\diamond$)	$(s \diamond t) \diamond u$	$=$	$s \diamond (t \diamond u)$	(degen')	$\{t\} \cup \{t\}$	$=$	t

Because of the first three lines, UnCAL has the cartesian products. For (c1), the proof is

$$\mathsf{cycle}(\mathsf{id}) =^{(\mathrm{unitL}\curlywedge)} \mathsf{cycle}(\curlywedge \diamond (\{\}_0 \times \mathsf{id})) =^{(\mathrm{nat}_Y)} \mathsf{cycle}(\curlywedge) \diamond \{\}_0 =^{(\mathrm{c2})} \mathsf{id} \diamond \{\}_0. = \{\}_0.$$

Lemma 2.2 *Under the assumption of Def. 2.1, the following is an UnCAL theorem.*

$$\text{(sub)} \qquad t \diamond \langle s_1, \cdots, s_k, \mathsf{id}_{Y'}\rangle = t\,[\vec{y} \mapsto \vec{s}]$$

3 Algebraic Semantics of UnCAL

In this section, we consider algebraic semantics of UnCAL. We also give a complete characterisation of the structural recursion, where e can depend on t in (★).

3.1 Iteration Σ-Algebras

We first review the notion of iteration Σ-algebras and various characterisation results by Bloom and Ésik. Let Σ be a signature, i.e. a set of function symbols equipped with arities. We define μ-terms by

$$t ::= x \mid f(t_1, \ldots, t_n) \mid \mu x.t,$$

where x is a variable. We use the convention that a function symbol $f^{(n)} \in \Sigma$ denotes n-ary. For a set V of variables, we denote by $\mathrm{T}(V)$ the set of all μ-terms generated by V. We define ConwayCI as the set of following equational axioms:

Conway equations

$$\mu x.t[s/x] = t[\mu x.\, s[t/x]/x],$$
$$\mu x.\mu y.t = \mu x.t[x/y]$$

Group equations associated with a group G

$$\mu x.(t[1 \cdot x/x], \ldots, t[n \cdot x/x])_1 = \mu y.(x[y/x], \ldots, [y/x])$$

Note that *the fixed point law*

$$\mu x.t = t[\mu x.t/x]$$

is an instance of the first axiom of Conway equations by taking $s = x$. The group equations [11] known as an alternative form of the commutative identities, are an axiom schema parameterised by a finite group $(G, \cdot)$ of order n, whose elements are natural numbers from 1 to n. We also note that the μ-notation is here extended on vectors $(t_1, \ldots, t_n)$, and $(-)_1$ denotes the first component of a vector. Given a vector $x = (x_1, \ldots, x_n)$ of distinct variables, the notation $i \cdot x = (x_{i\cdot 1}, \ldots, x_{i\cdot n})$ is used.

Definition 3.1 ([4]) A *pre-iteration* Σ*-algebra* $(A, (\!|-|\!)_A)$ consists of an nonempty set A and an interpretation function $(\!|-|\!)_A^{(-)} : \mathrm{T}(V) \times A^V \to A$ satisfying

(i) $(\!|x|\!)_A^\rho = \rho(x)$ for each $x \in V$ (iii) $(\!|t|\!)_A = (\!|t'|\!)_A \Longrightarrow (\!|\mu x.t|\!)_A = (\!|\mu x.t'|\!)_A$.

(ii) $(\!|t[t_1/x_1, \cdots, t_n/x_n]|\!)_A^\rho = (\!|t|\!)_A^{\rho'}$ with $\rho'(x_i) = (\!|t_i|\!)_A^\rho$, $\rho'(x) = \rho(x)$ for $x \neq x_i$

A pre-iteration Σ-algebra can be seen as a Σ-algebra $(A, \{f_A \mid f \in \Sigma\})$ with extra operations $(\!|\mu x.t|\!)_A$ for all t. A pre-iteration Σ-algebra A *satisfies* an equation $s = t$ over μ-terms, if $(\!|s|\!)_A = (\!|t|\!)_A$. Let E be a set of equations over μ-terms. An *iteration* Σ*-algebra* is a pre-iteration Σ-algebra that satisfies all equations in ConwayCI. An *iteration* (Σ, E)*-algebra* is an iteration Σ-algebra that satisfies all equations in E. A homomorphism of iteration Σ-algebras $h : A \to B$ is a function such that $h \circ (\!|t|\!)_A = (\!|t|\!) \circ h^V$ for all t. Since the variety of iteration Σ-algebras is exactly the variety of all continuous Σ-algebras ([4] Introduction), the interpretation of $\mu x.t$ in an iteration Σ-algebra can be determined through it.

We now regard each label $\ell \in L$ as an unary function symbol. Then we consider an iteration $L \cup \{0^{(0)}, +^{(2)}\}$-algebra. We define the axiom set AxBR by

$$\begin{array}{lll} s+(t+u) = (s+t)+u & s+t = t+s & t+0 = t \\ \mu x.x = 0 & \mu x.(x+y) = y & \text{for } y \text{ not containing } x \end{array}$$

and $\mathsf{AxCBR} \triangleq \mathsf{ConwayCI} \cup \mathsf{AxBR}$. We write $\mathsf{AxCBR} \vdash_\mu s = t$ if an equation $s = t$ is derivable from AxCBR by the standard equational logic EL-μ for μ-terms. For example, idempotency is derivable:

$$\mathsf{AxCBR} \vdash_\mu t + t = t$$

The proof is $t = \mu x.(x+t) = (\mu x.(x+t)) + t = t + t$, which uses the last axiom in AxBR and the fixed point law. Since μ-terms can be regarded as a representation of process terms of regular behavior as Milner shown in [23] (or synchronization trees [3]), the standard notion of strong bisimulation between two μ-terms can be defined. We write $s \sim t$ if they are bisimilar.

Theorem 3.2 ([3, 4, 12, 13])

(i) The axiom set AxCBR completely axiomatises the bisimulation, i.e., $\mathsf{AxCBR} \vdash_\mu s = t \iff s \sim t$

(ii) The set $\mathrm{T}(V)$ *of all* μ*-terms forms a free pre-iteration* Σ*-algebra over* V.

(iii) The set $\mathcal{BR}$ *of all regular* L*-labeled trees having* V*-leaves modulo bisimulation forms a free iteration* $(L \cup \{0, +\}, \mathsf{AxBR})$*-algebra over* V *([12] below Lemma 2, [24] Thm. 2).*

Note that $\mathcal{BR}$ stands for **R**egular trees modulo **B**isimulation, and AxBR stands for the axioms for regular trees modulo bisimulation.

3.2 Characterising UnCAL Normal Forms

UnCAL normal forms. Given an UnCAL term t of type &, we compute the *normal form* of t by the following three rewrite rules (N.B. we do not here use the other axioms) as a rewrite system [2], which are oriented equational axioms taken from the derived theory, AxGr and abbreviations.

$$\begin{array}{lrcl} \text{(sub)} & t \diamond \langle s_1, \cdots, s_k, \mathsf{id} \rangle & = & t\,[\vec{y} \mapsto \vec{s}] \\ \text{(Bekič)} & \mathsf{cycle}(\langle t, s \rangle) & = & \langle \pi_2, \mathsf{cycle}(s) \rangle \diamond \langle \mathsf{id}_A, \mathsf{cycle}(t \diamond \langle \mathsf{id}_{A \times V}, \mathsf{cycle}(s) \rangle) \rangle \\ \text{(union)} & \curlywedge \diamond (s, t) & = & \{s\} \cup \{t\} \end{array}$$

Let $\mathcal{M}$ be the set of all rewriting normal forms by the above rules, which finally erases all $\langle -, - \rangle$ and $\diamond$ in a given t. Normal forms are uniquely determined because the rewrite rules are confluent and terminating, hence have the unique normal form property [2]. Then by induction on terms we have that terms in $\mathcal{M}$ follow the grammar

$$\mathcal{M} \ni t ::= \quad y \quad | \quad \ell{:}\, t \quad | \quad \mathsf{cycle}^X(t) \quad | \quad \{\} \quad | \quad \{s\} \cup \{t\} \quad | \quad (x \triangleleft t).$$

Any outermost definition must be of the form $(\& \triangleleft t')$ by the assumption that the original given t is of type $\&$, thus we identity it with t'. Other definitions appear inside of t, as the following cases:

- Case $\{(x_1 \triangleleft t_1)\} \cup \{(x_2 \triangleleft t_2)\}$. We identify it with merely $\{t_1\} \cup \{t_2\}$, because marker names x_1, x_2 are hidden by this construction.
- Case $Y \vdash \mathsf{cycle}^x(x \triangleleft t') : x$. We identify it with merely $\mathsf{cycle}^{\&}(t')$, because these are equivalent by renaming of free maker x.

The *UnCAL normal forms* $\mathcal{N}$ are obtained from $\mathcal{M}$ by these identifications. It is of the form

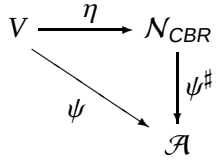
$$\begin{array}{lllllllll} \mathcal{N} \ni & t ::= & y & | & \ell{:}\, t & | & \mathsf{cycle}^X(t) & | & \{\} \quad | \quad \{s\} \cup \{t\} \\ \mathrm{T}(V) \ni & t ::= & y & | & \ell(t) & | & \mu x_1 \ldots \mu x_n . t & | & 0 \quad | \quad s + t \end{array}$$

Every normal form bijectively corresponds to a μ-term in $\mathrm{T}(V)$, i.e. $\mathcal{N} \cong \mathrm{T}(V)$, because each the above construct corresponds to the lower one, where $X = \langle\langle x_1, \ldots, x_n \rangle\rangle$. Hereafter, we may identify normal forms and μ-terms as above. Define the pair of signature and axioms by

$$\mathsf{UnC} \triangleq (L \cup \{0, +\}, \ \mathsf{AxBR}).$$

We regard an arbitrary UnC-algebra $\mathcal{A}$ as an *algebraic model* of UnCAL graphs. First, we show the existence of a free model. Define $\mathcal{N}_{\mathsf{CBR}}$ to be the quotient of $\mathcal{N}$ by the congruence generated by AxCBR.

Proposition 3.3
$\mathcal{N}_{CBR}$ forms a free iteration UnC-algebra over V. Thus for any function $\psi : V \to \mathcal{A}$, there exists an unique UnC-algebra homomorphism $\psi^\sharp$ such that the right diagram commutes, where η is an embedding of variables.



Proposition 3.4 $\mathcal{N}_{CBR} \cong \mathcal{BR}$.

Proof. By Theorem 3.2 (iii). □

3.3 Completeness of the Axioms for Bisimulation

Buneman et al. formulated that UnCAL graphs were identified by *extended bisimulation*, which is a bisimulation on graphs involving ε-edges. As discussed in §1, since our approach is to use only UnCAL terms, it suffices to consider only the standard (strong) bisimulation between UnCAL terms, as done in [23, 3, 12, 13]. We denote by $\sim$ bisimulation for UnCAL term.

In this subsection, we show the completeness of AxGr for bisimulation, using the following Lemma 3.5 that reduces the problem of $\mathsf{EL\text{-}UnCAL}$ to that of EL-μ through UnCAL normal forms. AxCBR has been shown to be complete for the bisimulation [3].

Lemma 3.5 *For UnCAL normal forms $n, m \in \mathcal{N}$, $\mathsf{AxCBR} \vdash_\mu n = m \iff Y \vdash n = m : X$ is derivable from AxGr in $\mathsf{EL\text{-}UnCAL}$.*

Proof. [⇒] : By induction on proofs of EL-μ. For every axiom in AxCBR, there exists the corresponding axiom in AxGr or an EL-UnCAL theorem, hence it can be emulated.
[⇐] : By induction on proofs of EL-UnCAL. Let $s = t$ is an axiom of EL-UnCAL. It easy to see that taking normal forms of both side, they are equal term, or correspond to an axiom in AxCBR or EL-μ theorem. □

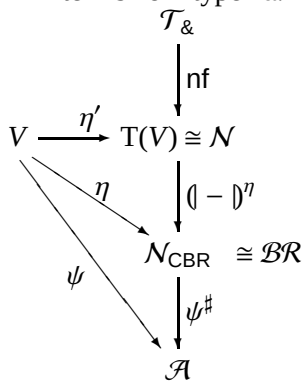
Theorem 3.6 (Completeness) *AxGr is sound and complete for the bisimulation, i.e.,*
$Y \vdash s = t \ : X$ *is derivable from AxGr in EL-UnCAL iff* $s \sim t$.

Proof. [⇒] : Because every axiom in AxGr is bisimilar, and the bisimulation is closed under contexts and substitutions [8].
[⇐] : Suppose $s \sim t$. Since for each rewrite rule for the normalisation function nf, both sides of the rule is bisimilar, nf preserves the bisimilarity. So we have $s \sim \mathsf{nf}(s) \sim \mathsf{nf}(t) \sim t$. Since AxCBR is complete axioms of bisimulation [3, 12], $\mathsf{AxCBR} \vdash_\mu \mathsf{nf}(s) = \mathsf{nf}(t)$. By Lemma 3.5, we have a theorem $Y \vdash \mathsf{nf}(s) = \mathsf{nf}(t) \ : X$. Thus $s = t$ is derivable. □

3.4 Interpretation in Algebraic Models

To interpret UnCAL terms and equations, we connect two freeness results in Thm. 3.2. Since UnCAL normal forms $\mathcal{N}$ is isomorphic to a free pre-iteration algebra $\mathrm{T}(V)$, it has the universal property. Define $\mathcal{T}_\&$ to be the set of all well-typed UnCAL terms of type &. We define $\mathsf{nf} : \mathcal{T}_\& \to \mathcal{N}$ by the function to compute the UnCAL normal form of a term. Then for any derivable equation $Y \vdash s = t \ : X$ in EL-UnCAL, we have $\mathsf{AxCBR} \vdash \mathsf{nf(s)} = \mathsf{nf(t)}$ by Lemma 3.5, thus for all assignment $\psi : V \to \mathcal{A}$,

$$\psi^\sharp(\!|\, \mathsf{nf}(s) \,|\!)^\eta = \psi^\sharp(\!|\, \mathsf{nf}(t) \,|\!)^\eta$$



where η and η' are embedding of variables.

Since $\mathcal{N}_{\mathsf{CBR}} \cong \mathcal{BR}$, we name the isomorphisms $\underline{(-)} : \mathcal{N}_{\mathsf{CBR}} \to \mathcal{BR}$ and $\overline{(-)} : \mathcal{BR} \to \mathcal{N}_{\mathsf{CBR}}$. We write simply a normal form t to denote a representative $[t]$ in $\mathcal{N}_{\mathsf{CBR}}$. Thus given a normal form t (which is a syntactic term, always finite), $\underline{t}$ is a (possibly infinite) regular tree by obtained by expanding cycles in t using fixpoints. Conversely, notice that since $\underline{t}$ is a tree, there are no cycles and the original cycles in t are infinitely expanded. Since $\mathcal{N} \cong \mathrm{T}(V)$, the functions $\underline{(-)}$ may also be applied to μ-terms. The iteration UnC-algebra $\mathcal{BR}$ has operations $0_{\mathcal{BR}} = \underline{\{\,\}}$, $+_{\mathcal{BR}}(r,s) = \underline{\overline{\{r\}} \cup \overline{\{s\}}}$, $\ell_{\mathcal{BR}}(r) = \underline{\ell(\overline{r})}$.

3.5 Deriving structural recursion of involved case

Next we model UnCAL's structural recursion of graphs. We use pairs of "the recursive computation" and the history of data structure. This is similar to the technique of paramorphism [22], which is a way to represent primitive recursion in terms of "fold" in functional programming. Our universal characterisation of graphs is the key to make this possible by the unique homomorphism from the free pre-iteration UnC-algebra $\mathcal{N}$ using the above analysis.

We take a term $X \vdash e_\ell(v,r) \ : X$ involving metavariables v and r, where $e_\ell(F(t),t)$ is the right-hand side e of $F(\ell\!:\!t)$ in (★) . For example, in case of the example `f1` in Introduction (see also Example 3.9), we take

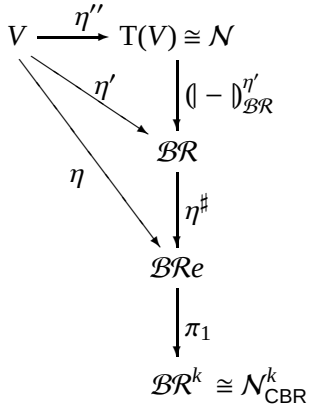
$$\begin{array}{lll} e_\ell(v,r) \triangleq \texttt{result:}r, & e_\ell(F(t),t) = \texttt{result:}t & \text{if } \ell = \texttt{ethnicGroup} \\ e_\ell(v,r) \triangleq v, & e_\ell(F(t),t) = F(t) & \text{if } \ell \neq \texttt{ethnicGroup} \end{array}$$

We construct a *specific* iteration UnC-algebra $\mathcal{BRe}$ for $\{e_\ell(v,r)\}_{\ell\in L}$. Let $k \triangleq |X|$. Without loss of generality, we can assume that $e_\ell(v,r)$ is of the form $\langle t_1, \cdots, t_k\rangle$ where every t_i is a normal form. We define the iteration UnC-algebra $\mathcal{BRe} = \mathcal{BR}^k \times \mathcal{BR}$ having operation

$$\ell_{\mathcal{BRe}}(v,r) = (\underline{e_\ell(v,r)},\ \underline{\ell\!:\!\overline{r}}), \qquad 0_{\mathcal{BRe}} = (\underline{\vec{\{\}}}, \underline{\{\}})$$

and $+_{\mathcal{BRe}}$ is an obvious tuple extensions of $+_{\mathcal{BR}}$. Here $\vec{\{\}}$ is the k-tuple of $\{\}$. Hereafter, we will use this convention $\vec{o}$ of tuple extension of an operator o.

Then, two freeness results in Thm. 3.2 are depicted in the right diagram, where $\eta(x) = (\underline{x_1}, \cdots, \underline{x_k}, \underline{x})$. Since $\mathrm{T}(V) \cong \mathcal{N}$, the interpretation in $\mathcal{BRe}$ is described as



$$(\!|x|\!)^\eta_{\mathcal{BRe}} = \eta(x), \quad (\!|\{\}|\!)^\eta_{\mathcal{BRe}} = 0_{\mathcal{BRe}}, \quad (\!|\{s\}\cup\{t\}|\!)^\eta_{\mathcal{BRe}} = (\!|s|\!)_{\mathcal{BRe}} +_{\mathcal{BRe}} (\!|t|\!)_{\mathcal{BRe}}$$
$$(\!|\ell\!:\!t|\!)^\eta_{\mathcal{BRe}} = \ell_{\mathcal{BRe}}((\!|t|\!)^\eta_{\mathcal{BRe}}), \quad (\!|\mathsf{cycle}(t)|\!)^\eta_{\mathcal{BRe}} = \eta^\sharp(\underline{\mathsf{cycle}(t)})$$

Now $(\!|-|\!)^\eta_{\mathcal{BRe}}$ is characterised as the unique pre-iteration $L\cup\{0,+\}$-algebra homomorphism from $\mathrm{T}(V)$ that extends η. Defining

$$\phi \triangleq \pi_1 \circ (\!|-|\!)^\eta_{\mathcal{BRe}} : \mathcal{N} \longrightarrow \mathcal{BR}^k \cong \mathcal{N}^k_{\mathsf{CBR}},$$

it is the unique function satisfying

$$\phi(x) = (\underline{x_1}, \ldots, \underline{x_k}), \qquad \phi(\{\}) = \underline{\vec{\{\}}}, \qquad \phi(\{s\}\cup\{t\}) = \underline{\phi(s)\ \vec{\cup}\ \phi(t)},$$
$$\phi(\ell\!:\!t) = \underline{e_\ell(v,t)}, \qquad \phi(\mathsf{cycle}(t)) = \pi_1 \circ \eta^\sharp(\underline{\mathsf{cycle}(t)})$$

The function ϕ takes normal forms of the type &. For non-normal forms, just precompose nf, i.e., define the function $\Phi : \mathcal{T}_\& \to \mathcal{N}^k_{\mathsf{CBR}}$ by $\Phi(s) \triangleq \phi(\mathsf{nf}(s))$, thus, $\Phi^{|X|} : \mathcal{T}_X \to {\mathcal{N}_{\mathsf{CBR}}}^{k|X|} \to \mathcal{T}^k_X$, because $\mathcal{T}_X \cong \mathcal{T}^{|X|}_\&$. In summary, we have the following, where s is a possibly non-normal form

$$\begin{array}{llll}
\Phi(s) & = \phi(\mathsf{nf}(s)) & \phi(x) & = \langle x_1, \cdots, x_k\rangle \qquad \phi(\{\}) = \vec{\{\}} \\
\Phi^{|X|+|Y|}(\langle t_1, t_2\rangle) & = \Phi^{|X|}(t_1) \vec{\times} \Phi^{|Y|}(t_2) & \phi(\ell\!:\!t) & = e_\ell(\phi(t), t) \qquad \phi(t_1 \cup t_2) = \phi(t_1)\ \vec{\cup}\ \phi(t_2) \qquad (2) \\
\Phi^0(()) & = () & \phi(\mathsf{cycle}(t)) & = \pi_1 \circ \eta^\sharp(\underline{\mathsf{cycle}(t)})
\end{array}$$

where $\vec{\times}$ is the "zip" operator of two tuples. Here we use a map $\mathcal{N}_{\mathsf{CBR}} \to \mathrm{Tm}(V)$ to regard a normal form modulo AxCBR as a term, for which any choise of representative is harmless, because UnCAL graphs are identified by bisimulation and AxCBR axiomatises it. Identifying three kinds functions $\Phi, \Phi^{|X|}, \phi$ as a single function (also denoted by Φ, by abuse of notion) on $\mathrm{Tm}(V)$, this Φ is essentially what Buneman et al. [8] called the structural recursion on graphs for the case that e depends on t. Actually, we could make the characterisation more precise than [8], i.e., we obtain also the laws for the cases of $\diamond$ (by the case $\Phi(s) = \phi(\mathsf{nf}(s))$) and cycle, which tells how to compute them.

This is not merely rephrasing the known result, but also a stronger characterisation, which gives precise understanding of the structural recursion on graphs:

(i) Buneman et al. stated that (1) without ($\bowtie$) is a *property* ([8] Prop. 3) of a "structural recursive function on graphs" defined by the algorithms in [8]. This property (i.e. soundness) is desirable, but unfortunately, no completeness was given. There may be many functions that satisfy the property. In contrast to it, our characterisation is sound and *complete*: (2) determines a *unique* function by the universality.

(ii) This derivation does not entail $\Phi(s \diamond t) = \Phi(s) \diamond \Phi(t)$. It tells us that the only way to compute $\Phi(s \diamond t)$ is to compute the normal form of $s \diamond t$ and then apply ϕ.

(iii) This analysis does not entail $\Phi(\mathsf{cycle}(t)) = \mathsf{cycle}(\Phi(t))$ either. The iteration algebra structure tells us that the homomorphism ϕ maps a term $\mathsf{cycle}(t)$ to its interpretation in $\mathcal{BR}e$ where the cycles are expanded in a regular tree and at the same time, labels ℓ are interpreted using the operations of $\mathcal{BR}e$.

(iv) The structure preserved by structural recursion is the *(pre-)iteration algebra structure*. The structural recursive function ϕ is the composition of a pre-iteration algebra homomorphism, an a iteration algebra homomorphism and a projection.

3.6 Examples

We may use the notation $\{t_1, t_2, \ldots\}$ as the abbreviation of $\{t_1\} \cup \{t_2\} \cup \cdots$.

Example 3.7 ([8] Replace all labels with a) This is the example considered in Introduction.

```
sfun f2(L:T) = a:f2(T)
```

In this case, the recursion does *not* depend on `T` (because the right-hand side uses merely `f2(T)`). We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\ell_{\mathcal{BR}e}(v, r) = (\mathtt{a}\!:\!v,\ \ell\!:\!r).$$

(We may omit over and underlines to denote the isomorphisms for simplicity). Then Φ is the desired structural recursive function `f2`. E.g.

$$\Phi(\mathtt{b}\!:\!\mathsf{cycle}(\mathtt{c}\!:\!\&)) = \mathtt{a}\!:\!\phi(\mathsf{cycle}(\mathtt{c}\!:\!\&)) = \mathtt{a}\!:\!\pi_1 \circ \eta^\sharp(\mathtt{c}\!:\!\mathtt{c}\!:\!\cdots) = \mathtt{a}\!:\!(\overline{\mathtt{a}\!:\!\mathtt{a}\!:\!\cdots}) = \mathtt{a}\!:\!\mathsf{cycle}(\mathtt{a}\!:\!\&)$$

Example 3.8 ([8] Double the children of each node)

```
sfun f4(L:T) = {a:f4(T)} ∪ {b:f4(T)}
```

Example of execution.

```
f4(a:b:c:{})
⇝ {a:{ a:{a:{}, b:{}}, b:{a:{}, b:{}} }} ∪ {b:{ a:{a:{}, b:{}}, b:{a:{}, b:{}} }}
```

This case does *not* depend on `T`. We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\ell_{\mathcal{BR}e}(v, r) = (\{\mathtt{a}\!:\!v\} \cup \{\mathtt{b}\!:\!v\},\ \ell\!:\!r).$$

Then Φ gives the structural recursive function defined by `f4`.

Example 3.9 ([8] Retrieve all ethnic groups) We revisit the example given in §1.
For the structural recursive recursive definition of `f1`,

```
sfun f1(L:T) = if L = ethnicGroup then (result:T) else f1(T)
```

This case *does* depend on `T`. Example of execution:

```
f1(sd) ⇝ {result:"Celtic":{}, result:"Portuguese":{}, result:"Italian":{}}
```

We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\texttt{ethnicGroup}_{\mathcal{BR}e}(v,r) \triangleq (\texttt{result:}r,\ \texttt{ethnicGroup:}r)$$
$$\ell_{\mathcal{BR}e}(v,r) \triangleq (v,\ \ell\texttt{:}r) \text{ for } \ell \neq \texttt{ethnicGroup}$$

Then Φ is the structural recursive function defined by `f1`:

```
Φ (sd) = {result:"Celtic":{}, result:"Portuguese":{}, result:"Italian":{}}
```

Example 3.10 Consider another example in §1 of `aa?`. This case *does* depend on `T`. We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\texttt{a}_{\mathcal{BR}e}(v,r) \triangleq (\texttt{a?}(r),\ \texttt{a:}r)$$
$$\ell_{\mathcal{BR}e}(v,r) \triangleq (v,\ \ell\texttt{:}r) \quad \text{for } \ell \neq \texttt{a}.$$

Then Φ gives the structural function `aa?`

$$\Phi(\texttt{(a:\&)@(a:\{\})}) = \phi(\,\mathsf{nf}(\texttt{(a:\&)@(a:\{\})})\,) = \phi(\texttt{a:a:\{\}}) = \texttt{true:\{\}}$$
$$\Phi(\texttt{cycle(a:\&)}) = \pi_1 \circ \eta^\sharp(\underline{\texttt{cycle(a:\&)}}) = \pi_1 \circ \eta^\sharp(\texttt{a:a:}\cdots) = \pi_1(\texttt{a?(a:}\cdots\texttt{),a:}\cdots) = \texttt{true} : \{\}$$

4 Conclusion

In this paper, we have shown an application of Bloom and Ésik's iteration algebras to model graph data used in UnQL/UnCAL for describing and manipulating graphs. We have formulated UnCAL and given an axiomatisation of UnCAL graphs that characterises the original bisimulation. We have given algebraic semantics using Bloom and Ésik's iteration iteration algebras. The main result of this paper was to show that completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. As a consequence, we have given a clean characterisation of the computation mechanism of UnCAL, called "structural recursion on graphs" using free iteration algebra.

Acknowledgments. I am grateful to Kazutaka Matsuda and Kazuyuki Asada for discussions about UnCAL and its interpretation, and their helpful comments on a draft of the paper. A part of this work was done while I was visiting National Institute of Informatics (NII) during 2013 – 2014.

References

[1] K. Asada, S. Hidaka, H. Kato, Z. Hu & K. Nakano (2013): *A parameterized graph transformation calculus for finite graphs with monadic branches.* In: *Proc. of PPDP '13*, pp. 73–84, doi:10.1145/2505879.2505903.

[2] F. Baader & T. Nipkow (1998): *Term Rewriting and All That.* Cambridge University Press.

[3] S. L. Bloom & Z. Ésik (1993): *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science, Springer.

[4] S. L. Bloom & Z. Ésik (1994): *Solving Polynomial Fixed Point Equations*. In: *Proc. of MFCS'94*, LNCS 841, pp. 52–67, doi:10.1007/3-540-58338-6_58.

[5] S. L. Bloom, Z. Ésik & D. Taubner (1993): *Iteration Theories of Synchronization Trees*. *Inf. Comput.* 102(1), pp. 1–55, doi:10.1006/inco.1993.1001.

[6] P. Buneman, S. Davidson, G. Hillebrand & D. Suciu (1996): *A query language and optimization techniques for unstructured data* . In: *Proc. of ACM-SIGMOD'96*, doi:10.1145/233269.233368.

[7] P. Buneman, S. B. Davidson, M. F. Fernandez & D. Suciu (1997): *Adding Structure to Unstructured Data*. In: *Proc. of ICDT '97*, pp. 336–350, doi:10.1007/3-540-62222-5_55.

[8] P. Buneman, M. F. Fernandez & D. Suciu (2000): *UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion*. *VLDB J.* 9(1), pp. 76–110, doi:10.1007/s007780050084.

[9] R.L. Crole (1993): *Categories for Types*. Cambridge Mathematical Textbook.

[10] Z. Ésik (1999): *Axiomatizing Iteration Categories*. *Acta Cybernetica* 14, pp. 65–82.

[11] Z. Ésik (1999): *Group Axioms for Iteration*. *Inf. Comput.* 148(2), pp. 131–180, doi:10.1006/inco.1998.2746.

[12] Z. Ésik (2000): *Axiomatizing the Least Fixed Point Operation and Binary Supremum*. In: *Proc. of Computer Science Logic 2000*, LNCS 1862, pp. 302–316, doi:10.1007/3-540-44622-2_20.

[13] Z. Ésik (2002): *Continuous Additive Algebras and Injective Simulations of Synchronization Trees*. *J. Log. Comput.* 12(2), pp. 271–300, doi:10.1093/logcom/12.2.271.

[14] M. P. Fiore & M. D. Campos (2013): *The Algebra of Directed Acyclic Graphs*. In: *Computation, Logic, Games, and Quantum Foundations*, LNCS 7860, pp. 37–51, doi:10.1007/978-3-642-38164-5_4.

[15] M. Hasegawa (1997): *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. Ph.D. thesis, University of Edinburgh. Distinguished Dissertation Series, Springer-Verlag, 1999.

[16] M. Hasegawa (1997): *Recursion from Cyclic Sharing: Traced Monoidal Categories and Models of Cyclic Lambda Calculi*. In: *Proc. of TLCA'97*, pp. 196–213, doi:10.1007/3-540-62688-3_37.

[17] S. Hidaka, K. Asada, Z. Hu, H. Kato & K. Nakano (2013): *Structural recursion for querying ordered graphs*. In: *Proc. of ACM SIGPLAN ICFP'13*, pp. 305–318, doi:10.1145/2500365.2500608.

[18] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda & K. Nakano (2010): *Bidirectionalizing graph transformations*. In: *Proc. of ICFP 2010*, pp. 205–216, doi:10.1145/1863543.1863573.

[19] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, K. Nakano & I. Sasano (2011): *Marker-Directed Optimization of UnCAL Graph Transformations*. In: *Proc. of LOPSTR'11*, pp. 123–138, doi:10.1007/978-3-642-32211-2_9.

[20] A. Joyal, R. Street & D. Verity (1996): *Traced monoidal categories*. *Mathematical Proceedings of the Cambridge Philosophical Society* 119(3), pp. 447–468, doi:10.1017/S0305004100074338.

[21] S. Mac Lane (1971): *Categories for the Working Mathematician*. *Graduate Texts in Mathematics* 5, Springer-Verlag, doi:10.1007/978-1-4612-9839-7.

[22] L. G. L. T. Meertens (1992): *Paramorphisms*. *Formal Asp. Comput.* 4(5), pp. 413–424, doi:10.1007/BF01211391.

[23] R. Milner (1984): *A Complete Inference System for a Class of Regular Behaviours*. *J. Comput. Syst. Sci.* 28(3), pp. 439–466, doi:10.1016/0022-0000(84)90023-0.

[24] P. M. Sewell (1995): *The Algebra of Finite State Processes*. Ph.D. thesis, University of Edinburgh. Dept. of Computer Science technical report CST-118-95, also published as LFCS-95-328.

[25] A. K. Simpson & G. D. Plotkin (2000): *Complete Axioms for Categorical Fixed-Point Operators*. In: *Proc. of LICS'00*, pp. 30–41, doi:10.1109/LICS.2000.855753.

Weak Completeness of Coalgebraic Dynamic Logics

Helle Hvid Hansen*
Delft University of Technology
Delft, The Netherlands
h.h.hansen@tudelft.nl

Clemens Kupke†
University of Strathclyde
Glasgow, United Kingdom
clemens.kupke@strath.ac.uk

We present a coalgebraic generalisation of Fischer and Ladner's Propositional Dynamic Logic (PDL) and Parikh's Game Logic (GL). In earlier work, we proved a generic strong completeness result for coalgebraic dynamic logics without iteration. The coalgebraic semantics of such programs is given by a monad T, and modalities are interpreted via a predicate lifting λ whose transpose is a monad morphism from T to the neighbourhood monad. In this paper, we show that if the monad T carries a complete semilattice structure, then we can define an iteration construct, and suitable notions of diamond-likeness and box-likeness of predicate-liftings which allows for the definition of an axiomatisation parametric in T, λ and a chosen set of pointwise program operations. As our main result, we show that if the pointwise operations are "negation-free" and Kleisli composition left-distributes over the induced join on Kleisli arrows, then this axiomatisation is weakly complete with respect to the class of standard models. As special instances, we recover the weak completeness of PDL and of dual-free Game Logic. As a modest new result we obtain completeness for dual-free GL extended with intersection (demonic choice) of games.

1 Introduction

Propositional Dynamic Logic (PDL) [4] and its close cousin Game Logic (GL) [14] are expressive, yet computationally well-behaved extensions of modal logics. Crucial for the increased expressiveness of these logics is the *-operator (iteration) that allows to compute certain, relatively simple fixpoint properties such as reachability or safety. This feature comes at a price: completeness proofs for deduction systems of logics with fixpoint operators are notoriously difficult. The paradigmatic example for this phenomenon is provided by the modal μ-calculus: Walukiewicz's completeness proof from [19] for Kozen's axiomatisation [10] is highly non-trivial and presently not widely understood.

Our main contribution is a completeness proof for coalgebraic dynamic logics *with iteration*. We introduced coalgebraic dynamic logics in our previous work [7] as a natural generalisation of PDL and GL with the aim to study various dynamic logics within a uniform framework that is parametric in the type of models under consideration, or - categorically speaking - parametric in a given monad. In [7] we presented an initial soundness and strong completeness result for such logics. Crucially, however, this only covered *iteration-free variants*. This paper provides an important next step by extending our previous work to the coalgebraic dynamic logic with iteration. As in the case of PDL, strong completeness fails, hence our coalgebraic dynamic logics with iteration are (only) proved weakly complete. While the concrete instances of our general completeness result are well-known [11, 14], the abstract coalgebraic nature of our proof allows us to provide a clear analysis of the general requirements needed for the PDL/GL completeness proof, leading to the notions of box- and diamond-like modalities and of a left-quantalic monad. As a modest new completeness result we obtain completeness for dual-free GL extended by intersection (demonic choice) of games.

*Supported by NWO-Veni grant 639.021.231.
†Supported by University of Strathclyde starter grant.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 90–104, doi:10.4204/EPTCS.191.9

At this relatively early stage of development our work has to be mainly regarded as a proof-of-concept result: we provide evidence for the claim that completeness proofs for so-called exogenous modal logics can be generalised to the coalgebraic level. This opens up a number of promising directions for future research which we will discuss in the Conclusion.

2 Coalgebraic Dynamic Logic

2.1 Coalgebraic modal logic

We assume some familiarity with the basic theory of coalgebra [16], monads and categories [13]. We start by recalling basic notions from coalgebraic modal logic, and fixing notation. For more information and background on coalgebraic modal logic, we refer to [12].

For a set X, we define $\mathsf{Prop}(X)$ to be the set of propositional formulas over X. Formally, $\mathsf{Prop}(X)$ is generated by the grammar: $\mathsf{Prop}(X) \ni \varphi ::= x \in X \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi$.

A *modal signature* Λ is a collection of modalities with associated arities. In this paper, we will only consider unary modalities. For a set X, we denote by $\Lambda(X)$ the set of expressions $\Lambda(X) = \{\Diamond x \mid \Diamond \in \Lambda\}$. The set $\mathscr{F}(\Lambda, P_0)$ of Λ-modal formulas over Λ and a set P_0 of atomic propositions is given by:

$$\mathscr{F}(\Lambda, P_0) \ni \varphi ::= p \in P_0 \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \qquad \Diamond \in \Lambda.$$

Let $T\colon \mathsf{Set} \to \mathsf{Set}$ be a functor. A *T-coalgebraic semantics* of $\mathscr{F}(\Lambda, P_0)$ is given by associating with each $\Diamond \in \Lambda$ a predicate lifting $\lambda\colon \mathscr{Q} \Rightarrow \mathscr{Q} \circ T$, where $\mathscr{Q}$ denotes the contravariant powerset functor. A *T-model* (X, γ, V) then consists of a carrier set X, a T-coalgebra $\gamma\colon X \to TX$, and a valuation $V\colon P_0 \to \mathscr{P}(X)$ that defines truth sets of atomic propositions as $[\![p]\!] = V(p)$. The truth sets of complex formulas is defined inductively as usual with the modal case given by: $[\![\Diamond\varphi]\!] = \gamma^{-1}(\lambda_X([\![\varphi]\!]))$.

A *modal logic* $\mathscr{L} = (\Lambda, \mathrm{Ax}, \mathrm{Fr}, \mathrm{Ru})$ consists of a modal signature Λ, a collection of rank-1 axioms $\mathrm{Ax} \subseteq \mathsf{Prop}(\Lambda(\mathsf{Prop}(P_0)))$, a collection $\mathrm{Fr} \subseteq \mathscr{F}(\Lambda, P_0)$ of frame conditions, and a collection of inference rules $\mathrm{Ru} \subseteq \mathscr{F}(\Lambda, P_0) \times \mathscr{F}(\Lambda, P_0)$ which contains the *congruence rule*: from $\varphi \leftrightarrow \psi$ infer $\Diamond\varphi \leftrightarrow \Diamond\psi$ for any modality $\Diamond \in \Lambda$.

Given a modal logic $\mathscr{L} = (\Lambda, \mathrm{Ax}, \mathrm{Fr}, \mathrm{Ru})$, the set of $\mathscr{L}$-derivable formulas is the smallest subset of $\mathscr{F}(\Lambda, P_0)$ that contains $\mathrm{Ax} \cup \mathrm{Fr}$, all propositional tautologies, is closed under modus ponens, uniform substitution and under applications of substitution instances of rules from Ru. For a formula $\varphi \in \mathscr{F}(\Lambda, P_0)$ we write $\vdash_{\mathscr{L}} \varphi$ if φ is $\mathscr{L}$-derivable. Furthermore φ is *$\mathscr{L}$-consistent* if $\nvdash_{\mathscr{L}} \neg\varphi$ and a finite set $\Phi \subseteq \mathscr{F}(\Lambda, P_0)$ is $\mathscr{L}$-consistent if the formula $\bigwedge\Phi$ is $\mathscr{L}$-consistent.

Next, we recall the following *one-step notions* from the theory of coalgebraic logic. Let X be a set.

- A formula $\varphi \in \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ is *one-step $\mathscr{L}$-derivable*, denoted $\vdash^1_{\mathscr{L}} \varphi$, if φ is propositionally entailed by the set $\{\psi\tau \mid \tau : P \to \mathscr{P}(X), \psi \in \mathrm{Ax}\}$.
- A set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ is called *one-step $\mathscr{L}$-consistent* if there are no formulas $\varphi_1, \ldots, \varphi_n \in \Phi$ such that $\vdash^1_{\mathscr{L}} \varphi_1 \wedge \cdots \wedge \varphi_n \to \bot$.
- Let T be a Set-functor and assume a predicate lifting $\lambda^{\Diamond}$ is given for each $\Diamond \in \Lambda$. For a formula $\varphi \in \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ the *one-step semantics* $[\![\varphi]\!]_1 \subseteq TX$ is defined by putting $[\![\Diamond(U)]\!]_1 = \lambda^{\Diamond}_X(U)$ and by inductively extending this definition to Boolean combinations of boxed formulas.
- For a set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ of formulas, we let $[\![\Phi]\!]_1 = \bigcap_{\varphi\in\Phi} [\![\varphi]\!]_1$, and we say that Φ is *one-step satisfiable* if $[\![\Phi]\!]_1 \neq \emptyset$.

- $\mathscr{L}$ is called *one-step sound* if for any one-step derivable formula $\varphi \in \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ we have $[\![\varphi]\!]_1 = TX$, i.e., if any such formula φ is *one-step valid*.
- $\mathscr{L}$ is called *one-step complete* if for every finite set X and every one-step consistent set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathscr{P}(X)))$ is one-step satisfiable.

2.2 Dynamic syntax and semantics

In earlier work [7], we introduced the notion of a coalgebraic dynamic logic for programs built from Kleisli composition, pointwise operations and tests. Here we extend this notion to also include iteration (Kleene star).

Throughout, we fix a countable set P_0 of atomic propositions, a countable set A_0 of atomic actions, and a signature Σ (of pointwise operations such as $\cup$ in PDL). The set $\mathscr{F}(P_0,A_0,\Sigma)$ of *dynamic formulas* and the set $A = A(P_0,A_0,\Sigma)$ of *complex actions* are defined by mutual induction:

$$\begin{array}{rcl} \mathscr{F}(P_0,A_0,\Sigma) \ni \varphi & ::= & p \in P_0 \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\alpha\rangle\varphi \\ A(P_0,A_0,\Sigma) \ni \alpha & ::= & a \in A_0 \mid \alpha;\alpha \mid \underline{\sigma}(\alpha_1,\dots,\alpha_n) \mid \alpha^* \mid \varphi? \end{array}$$

where $\underline{\sigma} \in \Sigma$ is n-ary.

Dynamic formulas are interpreted in dynamic structures which consist of a T-coalgebraic semantics with additional structure. Operation symbols $\underline{\sigma} \in \Sigma$ will be interpreted by pointwise defined operations on $(TX)^X$ induced by natural operations $\sigma\colon T^n \Rightarrow T$. More precisely, if $\sigma\colon T^n \Rightarrow T$ is a natural transformation, then $\sigma^X_X\colon ((TX)^X)^n \to (TX)^X$ is defined by $\sigma^X_X(f_1,\dots,f_n)(x) = \sigma_X(f_1(x),\dots,f_n(x))$. A natural transformation $\Sigma T \Rightarrow T$ (when viewing Σ as a Set-functor) corresponds to a collection of natural operations $\sigma\colon T^n \Rightarrow T$, one for each $\underline{\sigma} \in \Sigma$.

In order to define composition and tests of actions/programs/games, T must be a monad (T,μ,η) such that action composition amounts to Kleisli composition for T. In order to define iteration of programs, we need to assume that the monad has the following property.

Definition 2.1 (Left-quantalic monad) A monad (T,μ,η) is called *left-quantalic* if for all sets X, TX can be equipped with a sup-lattice structure (i.e., a complete, idempotent, join semilattice). We denote the empty join in TX by $\bot_{TX}$. We also require that when this join is lifted pointwise to the Kleisli Hom-sets $\mathscr{K}\ell(T)(X,X)$, then Kleisli-composition left-distributes over joins:

$$\forall f,g_i\colon X \to TX, i \in I\colon \quad f * \bigvee_i g_i = \bigvee_i f * g_i. \qquad \triangleleft$$

It is well known that Eilenberg-Moore algebras of the powerset monad $\mathscr{P}$ are essentially sup-lattices, and that relation composition left-distributes over unions of relations, hence $\mathscr{P}$ is left-quantalic. We observe that one way of showing that T is left-quantalic is to show that there is a morphism of monads $\tau\colon \mathscr{P} \Rightarrow T$.

Lemma 2.2 *Let (T,μ,η) be a monad. If there is a monad morphism $\tau\colon \mathscr{P} \Rightarrow T$, then (T,μ,η) is left-quantalic.*

Proof. A monad morphism $\tau\colon \mathscr{P} \Rightarrow T$ induces a functor $\mathscr{EM}(T) \to \mathscr{EM}(\mathscr{P})$ by pre-composition. It follows, in particular, that the free T-algebra is mapped to a sup-lattice $(TX, \mu_X \circ \tau_{TX})$. We extend this sup-lattice structure on TX pointwise to a sup-lattice structure on $\mathscr{K}\ell(T)(X,X)$, that is, for all $\{g_i \mid i \in I\} \subseteq \mathscr{K}\ell(T)(X,X)$,

$$(\bigvee_i g_i)(x) = \mu_X(\tau_{TX}(\{g_i(x) \mid i \in I\})).$$

Kleisli-composition distributes over this τ-induced join since μ_X and Tf preserve it, for all functions $f\colon X \to Y$, due to naturality of τ, and these maps being T-algebra morphisms. QED

Note that any natural transformation $\tau\colon \mathscr{P} \Rightarrow T$ yields a natural transformation $1 \Rightarrow \mathscr{P} \Rightarrow T$, where $1 \Rightarrow \mathscr{P}$ picks out the empty set, such that T is pointed as defined in [7].

Example 2.3 *The three monads of particular interest to us were described in [7]: The powerset monad $\mathscr{P}$, the monotone neighbourhood monad $\mathscr{M}$, the neighbourhood monad $\mathscr{N}$. These are all left-quantalic. For example, the transpose of the Kripke box $\widehat{\Box} = \tau_X\colon \mathscr{P}X \to \mathscr{M}X$ defined by $\tau_X(U) = \{V \subseteq X \mid U \subseteq V\}$ is a monad morphism. The join on $\mathscr{M}X$ induced by $\widehat{\Box}$ is intersection of neighbourhood collections. Dually, the transpose of the Kripke diamond $\widehat{\Diamond}_X(U) = \{V \subseteq X \mid U \cap V \neq \emptyset\}$ is also a monad morphism $\mathscr{P} \Rightarrow \mathscr{M}$, and its induced join is unions of neighbourhood collections.*

The generalisation of iteration for PDL-programs and GL-games is iterated Kleisli composition. Given $f\colon X \to TX$, we define for all $n < \omega$:

$$f^{[0]} = \eta_X, \qquad f^{[n+1]} = f * f^{[n]}, \qquad f^* = \bigvee_{n<\omega} f^{[n]} \tag{1}$$

Definition 2.4 (Dynamic semantics) Let $\mathbb{T} = (T, \eta, \mu)$ be a left-quantalic monad, and $\theta\colon \Sigma T \Rightarrow T$ a natural Σ-algebra. A (P_0, A_0, θ)-*dynamic* $\mathbb{T}$-*model* $\mathfrak{M} = (X, \gamma_0, \lambda, V)$ consists of a set X, an interpretation of atomic actions $\widehat{\gamma_0}\colon A_0 \to (TX)^X$, a unary predicate lifting $\lambda\colon \mathscr{Q} \Rightarrow \mathscr{Q} \circ T$ whose transpose $\widehat{\lambda}\colon T \Rightarrow \mathscr{N}$ is a monad morphism, and a valuation $V\colon P_0 \to \mathscr{P}(X)$. We define the truth set $[\![\varphi]\!]^{\mathfrak{M}}$ of dynamic formulas and the semantics $\widehat{\gamma}\colon A \to (TX)^X$ of complex actions in $\mathfrak{M}$ by mutual induction:

$$\begin{array}{lcll}
[\![p]\!]^{\mathfrak{M}} = V(p), & [\![\varphi \wedge \psi]\!]^{\mathfrak{M}} = [\![\varphi]\!]^{\mathfrak{M}} \cap [\![\psi]\!]^{\mathfrak{M}}, & [\![\neg\varphi]\!]^{\mathfrak{M}} = X \setminus [\![\varphi]\!]^{\mathfrak{M}}, & \\
[\![\langle\alpha\rangle\varphi]\!]^{\mathfrak{M}} & = & (\widehat{\gamma}(\alpha)^{-1} \circ \lambda_X)([\![\varphi]\!]^{\mathfrak{M}}), & \\
\widehat{\gamma}(\underline{\sigma}(\alpha_1, \ldots, \sigma_n)) & = & \sigma_X^X(\widehat{\gamma}(\alpha_1), \ldots, \widehat{\gamma}(\alpha_n)) & \text{where } \underline{\sigma} \in \Sigma \text{ is } n\text{-ary}, \\
\widehat{\gamma}(\alpha;\beta) & = & \widehat{\gamma}(\alpha) * \widehat{\gamma}(\beta) & \text{(Kleisli composition)}, \\
\widehat{\gamma}(\alpha^*) & = & \widehat{\gamma}(\alpha)^* & \text{(Kleisli iteration)}, \\
\widehat{\gamma}(\varphi?)(x) & = & \eta_X(x) \text{ if } x \in [\![\varphi]\!]^{\mathfrak{M}}, \perp_{TX} \text{ otherwise}. &
\end{array}$$

We say that $\mathfrak{M}$ validates a formula φ if $[\![\varphi]\!]^{\mathfrak{M}} = X$. A coalgebra $\gamma\colon X \to (TX)^A$ is *standard* if it is generated by some $\widehat{\gamma_0}\colon A_0 \to (TX)^X$ and $V\colon P_0 \to \mathscr{P}(X)$ as above, and we will also refer to (X, γ, λ, V) as a θ-dynamic $\mathbb{T}$-model. ◁

Recall that PDL can be axiomatised using the box or using the diamond, but the two axiomatisations differ. For example, the axioms for tests depend on which modality is used. In the general setting we need to know whether a predicate lifting corresponds to a box or a diamond.

Definition 2.5 (Diamond-like, Box-like) Let $\lambda\colon \mathscr{Q} \Rightarrow \mathscr{Q} \circ T$ be a predicate lifting for a left-quantalic monad T. We say that

- λ is *diamond-like* if for all sets X, all $U \subseteq X$, and all $\{t_i \mid i \in I\} \subseteq TX$:

$$\bigvee_{i\in I} t_i \in \lambda_X(U) \quad \text{iff} \quad \exists i \in I\colon t_i \in \lambda_X(U).$$

- λ is *box-like* if for all sets X, all $U \subseteq X$, and all $\{t_i \mid i \in I\} \subseteq TX$:

$$\bigvee_{i\in I} t_i \in \lambda_X(U) \quad \text{iff} \quad \forall i \in I\colon t_i \in \lambda_X(U).$$

◁

Remark 2.6 *Note that λ is diamond-like iff $\lambda_X(U)$ is a complete filter of the semilattice TX for all $U \subseteq X$. One also easily verifies that λ is diamond-like iff its Boolean dual is box-like. It is easy to see that if λ is diamond-like then it is also diamond-like according to our "old" definition in [7], similarly for box-like. However, it is no longer the case that every predicate lifting is either box-like or diamond-like, e.g., for $T = \mathscr{P}$, $\lambda_X(U) = \{V \subseteq X \mid \emptyset \neq V \subseteq U\}$ is neither.*

Example 2.7 *It can easily be verified that the Kripke diamond (box) is indeed diamond-like (box-like) for $\mathscr{P}$. Taking $T = \mathscr{M}$, and union as join on $\mathscr{M}X$ (i.e., the join induced by $\Diamond$, cf. Example 2.3), then the monotonic neighbourhood modality $\lambda_X(U) = \{N \in \mathscr{M}X \mid U \in N\}$ is diamond-like, but taking intersection as the join on $\mathscr{M}X$ then λ is box-like. Similarly, λ is diamond-like when viewed as a neighbourhood modality for $\mathscr{N}$-coalgebras with union as join. Note that this shows that diamond-likeness does not imply monotonicity. We only have, if λ is diamond-like, then $\widehat{\lambda} : T \Rightarrow \mathscr{N}$ is monotone.*

We will use the following crucial lemma about the Kleisli composition and predicate liftings.

Lemma 2.8 *Let $\lambda : \mathscr{Q} \Rightarrow \mathscr{Q} \circ T$ be a predicate lifting whose transpose $\widehat{\lambda} : T \Rightarrow \mathscr{N}$ is a monad morphism. For all $f, g : X \to TX$, all $x \in X$ and all $U \subseteq X$, we have*

$$(f * g)(x) \in \lambda_X(U) \iff f(x) \in \lambda_X(g^{-1}(\lambda_X(U))).$$

Proof. We have:

$$\begin{array}{rll}
(f*g)(x) \in \lambda_X(U) & \text{iff} & \mu_X(Tg(f(x))) \in \lambda_X(U) \\
(\text{def. of } \hat{\lambda}) & \text{iff} & U \in \hat{\lambda}_X(\mu_X(Tg(f(x)))) \\
(\hat{\lambda} \text{ monad morph.}) & \text{iff} & U \in \mu_X^{\mathscr{N}}\left(\mathscr{N}\hat{\lambda}_X(\hat{\lambda}_{TX}(Tg(f(x))))\right) \\
(\text{def. of } \mu^N) & \text{iff} & \eta_{\mathscr{P}(X)}(U) \in \mathscr{N}\hat{\lambda}_X\left(\hat{\lambda}_{TX}(Tg(f(x)))\right) \\
(\text{def. of } \mathscr{N}) & \text{iff} & \hat{\lambda}_X^{-1}\left(\eta_{\mathscr{P}(X)}(U)\right) \in \hat{\lambda}_{TX}(Tg(f(x))) \\
(\text{def. of } \eta) & \text{iff} & \{t \in TX \mid U \in \widehat{\lambda}_X(t)\} \in \hat{\lambda}_{TX}(Tg(f(x))) \\
(\text{def. of } \hat{\lambda}) & \text{iff} & \{t \in TX \mid t \in \lambda_X(U)\} \in \hat{\lambda}_{TS}(Tg(f(x))) \\
(\text{naturality of } \hat{\lambda}) & \text{iff} & \{t \in TX \mid t \in \lambda_X(U)\} \in \mathscr{N}g(\hat{\lambda}_X(f(x))) \\
(\text{def. of } \mathscr{N}) & \text{iff} & g^{-1}(\lambda_X(U)) \in \hat{\lambda}_X(f(x)) \\
 & \text{iff} & f(x) \in \lambda_X(g^{-1}(\lambda_X(U)))
\end{array}$$

QED

2.3 Coalgebraic dynamic logic

Our notion of a coalgebraic dynamic logic relates to coalgebraic modal logic in the same way that PDL relates to the basic modal logic **K**. In the remainder of the paper, we assume that:

- $\mathbb{T} = (T, \mu, \eta)$ is a left-quantalic monad with join $\bigvee : \mathscr{P}TX \to TX$,
- $\lambda : \mathscr{Q} \Rightarrow \mathscr{Q} \circ T$ is a diamond-like with respect to $(TX, \bigvee)$, monotonic predicate lifting whose transpose $\widehat{\lambda} : T \Rightarrow \mathscr{N}$ is a monad morphism,
- Σ is a signature and for each n-ary $\underline{\sigma} \in \Sigma$ there is a natural operation $\sigma : T^n \Rightarrow T$ and a natural operation $\chi : \mathscr{N}^n \Rightarrow \mathscr{N}$ such that $\widehat{\lambda} \circ \sigma = \chi \circ \widehat{\lambda}^n$. We denote by θ the collection $\{\sigma \mid \underline{\sigma} \in \Sigma\}$.

Using the last item above, we showed in [7, section 4] how to associate to each operation symbol $\underline{\sigma} \in \Sigma$ a rank-1 axiom $\langle\underline{\sigma}(\alpha_1, \ldots, \alpha_n)\rangle p \leftrightarrow \varphi(\check{\chi}, \alpha_1, \ldots, \alpha_n, p)$. Briefly stated, we use that a $\chi : \mathscr{N}^n \Rightarrow \mathscr{N}$ corresponds (via the Yoneda lemma) to an element $\check{\chi}$ of the free Boolean algebra $\mathscr{N}(n \cdot \mathscr{Q}(2))$ generated by $n \cdot \mathscr{Q}(2)$. By assigning a rank-1 formula to each of the generators, we obtain a rank-1 formula $\varphi(\check{\chi}, \alpha_1, \ldots, \alpha_n, p)$ for each χ. For example, the PDL axiom $\langle\alpha \cup \beta\rangle p \leftrightarrow \langle\alpha\rangle p \vee \langle\beta\rangle p$ is of this kind. Our completeness result will be restricted to positive operations.

Definition 2.9 (Positive natural operations) We call $\chi\colon \mathscr{N}^n \Rightarrow \mathscr{N}$ a *positive operation* if $\check{\chi}$ can be constructed using only $\wedge$ and $\vee$ in $\mathscr{N}(n\cdot\mathscr{Q}(2))$. If $\sigma\colon T^n \Rightarrow T$ and $\chi\colon \mathscr{N}^n \Rightarrow \mathscr{N}$ are such that $\widehat{\lambda}\circ\sigma = \chi\circ\widehat{\lambda}^n$, then we call σ positive if χ is positive. The axioms for positive pointwise operations of the form $\check{\chi} = \check{\delta}\wedge\check{\rho}$ are obtained by extending Definition 14 from [7] with a case for conjunction:

$$\varphi(\check{\delta}\wedge\check{\rho},\alpha_1,\ldots,\alpha_n,p) = \varphi(\check{\delta},\alpha_1,\ldots,\alpha_n,p)\wedge\varphi(\check{\rho},\alpha_1,\ldots,\alpha_n,p). \qquad \triangleleft$$

Example 2.10 *Positive natural operations on $\mathscr{P}$ include union, but complement and intersection are not natural on $\mathscr{P}$. Positive natural operations on $\mathscr{M}$ include union and intersection, but not the natural operation dual.*

Definition 2.11 (Dynamic logic) Let $\mathscr{L}_\Diamond = (\{\Diamond\},\mathrm{Ax},\emptyset,\mathrm{Ru})$ be a modal logic over the basic modal language $\mathscr{F}(\{\Diamond\},P_0)$. We define $\Lambda = \{\langle\alpha\rangle \mid \alpha\in A\}$ and let $\mathrm{Ax}_A = \bigcup_{\alpha\in A}\mathrm{Ax}_\alpha$ where Ax_α is the set of rank-1 axioms over the labelled modal language $\mathscr{F}(P_0,A_0,\Sigma)$ obtained by substituting $\langle\alpha\rangle$ for $\Diamond$ in all the axioms in Ax. We define Ru_A similarly as all labelled instances of rules in Ru.

The *θ-dynamic logic* over $\mathscr{L}_\Diamond$ is the modal logic $\mathscr{L} = \mathscr{L}(\theta,;,^*,?) = (\Lambda,\mathrm{Ax}',\mathrm{Fr}',\mathrm{Ru}')$ where

$$\begin{array}{rcl}
\mathrm{Ax}' & = & \mathrm{Ax}_A \cup \{\langle\underline{\sigma}(\alpha_1,\ldots,\alpha_n)\rangle p \leftrightarrow \varphi(\check{\chi},\alpha_1,\ldots,\alpha_n,p) \mid \underline{\sigma}\in\Sigma,\alpha_i\in A\} \\
\mathrm{Fr}' & = & \{\langle\alpha;\beta\rangle p \leftrightarrow \langle\alpha\rangle\langle\beta\rangle p \mid \alpha,\beta\in A, p\in P_0\}\cup \\
 & & \{\langle\alpha^*\rangle p \leftrightarrow p\vee\langle\alpha\rangle\langle\alpha^*\rangle p \mid \alpha\in A\}\cup \\
 & & \{\langle\psi?\rangle p \leftrightarrow (\psi\wedge p) \mid \psi\in\mathscr{F}(P_0,A_0,\Sigma)\} \\
\mathrm{Ru}' & = & \mathrm{Ru}_A \cup \left\{ \dfrac{\langle\alpha\rangle\psi\vee\varphi\to\psi}{\langle\alpha^*\rangle\varphi\to\psi} \mid \alpha\in A\right\}
\end{array} \qquad \triangleleft$$

Proposition 2.12 *If $\mathscr{L}_\Diamond$ is sound wrt to the T-coalgebraic semantics then the θ-*dynamic logic $\mathscr{L}$ *is sound wrt to the class of all θ-dynamic $\mathbb{T}$-models. In other words, for all $\varphi\in\mathscr{F}(P_0,A_0,\Sigma)$ and all θ-dynamic $\mathbb{T}$-models $\mathfrak{M} = (X,\gamma_0,\lambda,V)$ we have*

$$\vdash_{\mathscr{L}}\varphi \quad \textit{implies that} \quad \mathfrak{M} \textit{ validates } \varphi.$$

Proof. In [7], we showed soundness of the axioms for pointwise operations, sequential composition and tests with respect to θ-dynamic $\mathbb{T}$-models (without iteration). Soundness of the star axiom is not difficult to check. Soundness of the star rule can be proven as follows: Suppose $\mathfrak{M} = (X,\gamma,\lambda,V)$ is a θ-dynamic T-model such that $\mathfrak{M}$ validates the formula $\langle\alpha\rangle\psi\vee\varphi\to\psi$. For any state $x\in X$ such that $x\models\langle\alpha^*\rangle\varphi$ we have — by standardness of γ — that $\widehat{\gamma}(\alpha)^*(x)\in\lambda_X([\![\varphi]\!])$. This implies $\bigvee_j\widehat{\gamma}(\alpha)^{[j]}(x)\in\lambda_X([\![\varphi]\!])$ and, by diamond-likeness of λ, there is a $j\geq 0$ such that $\widehat{\gamma}(\alpha)^{[j]}(x)\in\lambda_X([\![\varphi]\!])$. Therefore, to show that $\mathfrak{M}$ validates $\langle\alpha^*\rangle\varphi\to\psi$, it suffices to show that for all $j\geq 0$ we have $U_j\subseteq[\![\psi]\!]$ where

$$U_j = \{x\in X \mid \widehat{\gamma}(\alpha)^{[j]}(x)\in\lambda_X([\![\varphi]\!])\}.$$

We prove this by induction. For $j=0$ the claim holds trivially as by assumption the premiss of the star rule is valid and thus $[\![\varphi]\!]\subseteq[\![\psi]\!]$. Consider now some $j=i+1$. Then we have

$$\begin{array}{rcll}
U_{i+1} & = & \{x\in X \mid \widehat{\gamma}(\alpha)^{[i+1]}(x)\in\lambda_X([\![\varphi]\!])\} & \\
 & = & \{x\in X \mid \widehat{\gamma}(\alpha)*\widehat{\gamma}(\alpha)^{[i]}(x)\in\lambda_X([\![\varphi]\!])\} & \\
 & \overset{\text{Lemma 2.8}}{=} & \{x\in X \mid \widehat{\gamma}(\alpha)(x)\in\lambda_X(U_i)\} & \\
 & \overset{\text{I.H.}}{\subseteq} & \{x\in X \mid \widehat{\gamma}(\alpha)(x)\in\lambda_X([\![\psi]\!])\} & \\
 & = & [\![\langle\alpha\rangle\psi]\!]\subseteq[\![\psi]\!] & \text{(last inclusion holds by validity of rule premiss)}
\end{array}$$

QED

3 Weak Completeness

In this section, we will show that if the base logic $\mathscr{L}_\Diamond$ is one-step complete with respect to the T-coalgebraic semantics given by λ, and θ consists of positive operations, then the dynamic logic $\mathscr{L} = \mathscr{L}(\theta,;,^*,?)$ is (weakly) complete with respect to the class of all θ-dynamic $\mathbb{T}$-models, i.e., every $\mathscr{L}$-consistent formula is satisfiable in a θ-dynamic $\mathbb{T}$-model. As in the completeness proof for PDL, a satisfying model for a formula ψ will essentially be obtained from a filtration of the canonical model through a suitable closure of $\{\psi\}$.

A set $\Phi \subseteq \mathscr{F}(P_0,A_0,\Sigma)$ of dynamic formulas is *(Fischer-Ladner) closed* if it is closed under subformulas, closed under single negation, that is, if $\varphi = \neg\psi \in \Phi$ then $\psi \in \Phi$, and if $\varphi \in \Phi$ is not a negation, then $\neg\varphi \in \Phi$, and satisfies the following closure conditions:

1. If $\langle\alpha;\beta\rangle\varphi \in \Phi$ then $\langle\alpha\rangle\langle\beta\rangle\varphi \in \Phi$.
2. For all 1-step axioms $\langle\underline{\sigma}(\alpha_1,\dots,\alpha_n)\rangle p \leftrightarrow \varphi(\check{\chi},\alpha_1,\dots,\alpha_n,p)$, if $\langle\underline{\sigma}(\alpha_1,\dots,\alpha_n)\rangle\psi \in \Phi$ then also $\varphi(\check{\chi},\alpha_1,\dots,\alpha_n,\psi) \in \Phi$.
3. If $\langle\psi?\rangle\varphi \in \Phi$ then $\psi \wedge \varphi \in \Phi$.
4. If $\langle\alpha^*\rangle\varphi \in \Phi$ then $\langle\alpha\rangle\langle\alpha^*\rangle\varphi$ and $\langle\alpha\rangle\varphi \in \Phi$.

Given a dynamic formula ψ, we denote by $Cl(\psi)$ the least set of formulas that is closed and contains ψ. A standard argument shows that $Cl(\psi)$ is finite.

From now on we fix a finite, closed set Φ (which may be thought of as $Cl(\psi)$ for some ψ). An *$\mathscr{L}$-atom over Φ* is a maximally $\mathscr{L}$-consistent subset of Φ, and we denote by S the set of all $\mathscr{L}$-atoms over Φ. For $\varphi \in \mathscr{F}(P_0,A_0,\Sigma)$ we put $\hat{\varphi} = \{\Delta \in S \mid \varphi \in \Delta\}$.
Note that, in particular, for each $\varphi \notin \Phi$ we have $\hat{\varphi} = \emptyset$. A maximally $\mathscr{L}$-consistent set (MCS) Ξ is a maximally $\mathscr{L}$-consistent subset of $\mathscr{F}(P_0,A_0,\Sigma)$. Clearly, for each MCS Ξ we have $\Xi \cap \Phi$ is an $\mathscr{L}$-atom. Any subset of S can be characterised by a propositional combination of formulas in Φ. It will be useful to have a notation for these characteristic formulas at hand.

Definition 3.1 (Characteristic formula) For $U \subseteq S$, we define the characteristic formula ξ_U of U by

$$\xi_U = \bigvee_{\Delta \in U} \bigwedge \Delta$$

where for any $\Delta \in S$, $\bigwedge\Delta$ is the conjunction of the elements of Δ. ◁

We will use the following fact that allows to lift one-step completeness of the base logic to $\mathscr{L}$.

Lemma 3.2 *If $\mathscr{L}_\Diamond$ is one-step complete for T then $\mathscr{L}$ is one-step complete for T^A.*

The proof of this lemma is analogous to the proof of the corresponding statement in [6]. The main difference being that instead of arguing via MCSs one has to use atoms. Note that only the axioms for pointwise operations have influence on one-step properties, as the ones for ; and * are not rank-1.

3.1 Strongly coherent models

As in the finitary completeness proof of PDL [11] and the finite model construction in [18], we need a coalgebra structure on the set S of all $\mathscr{L}$-atoms over Φ that satisfies a certain coherence condition which ensures that a truth lemma can be proved.

Definition 3.3 (Coherent structure) A coalgebra $\gamma\colon S \to (TS)^A$ is *coherent* if for all $\Gamma \in S$ and all $\langle\alpha\rangle\varphi \in \Phi$, $\quad \widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(\hat{\varphi}) \quad$ iff $\quad \langle\alpha\rangle\varphi \in \Gamma$. $\triangleleft$

Lemma 3.4 (Truth lemma) *Let $\gamma\colon S \to (TS)^A$ be a coherent structure map and define a valuation $V : P_0 \to \mathscr{P}(S)$ for propositional variables $p \in P_0$ by putting $V(p) = \hat{p}$. For each $\Gamma \in S$ and $\varphi \in \Phi$ we have*

$$(S,\gamma,V),\Gamma \models \varphi \qquad \textit{iff} \qquad \varphi \in \Gamma.$$

The lemma follows from a standard induction argument on the structure of the formula φ - the base case is a immediate consequence of the definition of the valuation, the induction step for the modal operators follows from coherence.

In order to prove coherence for iteration programs α^*, we need the following stronger form of coherence, which is inspired by the completeness proof of dual-free Game Logic in [14].

Definition 3.5 (Strongly coherent structure) We say that $\gamma\colon S \to (TS)^A$ is *strongly coherent for $\alpha \in A$* if for all $\Gamma \in S$ and all $U \subseteq S$: $\quad \widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(U) \quad$ iff $\quad \langle\alpha\rangle\xi_U \wedge \Gamma$ is $\mathscr{L}$-consistent. $\triangleleft$

In the remainder of this subsection, we prove the following existence result.

Proposition 3.6 *If $\mathscr{L}_\Diamond$ is one-step complete for T, then there exists a $\gamma\colon S \to (TS)^A$ which is strongly coherent for all $\alpha \in A$.*

Let $(-)^\sharp\colon \mathsf{Prop}(\Lambda(\mathscr{P}(S))) \to \mathsf{Prop}(\Lambda(\mathsf{Prop}(\Phi)))$ be the substitution map induced by taking $U^\sharp = \xi_U$ for all $U \in \mathscr{P}(S)$. Conversely, let $(-)_S\colon \mathsf{Prop}(\Lambda(\mathsf{Prop}(\Phi))) \to \mathsf{Prop}(\Lambda(\mathscr{P}(S)))$ be the substitution map induced by taking $\top_S = S$ and for all $\psi \in \mathsf{Prop}(\Phi)$, $\psi_S = \{\Delta \in S \mid \Delta \vdash_{PL} \psi\}$.

Lemma 3.7 (Derivability) *For all $\varphi \in \mathsf{Prop}(\Lambda(\mathsf{Prop}(\Phi)))$,*

1. $\vdash^1_{\mathscr{L}} \varphi_S \quad$ *implies* $\quad \vdash_{\mathscr{L}} (\varphi_S)^\sharp$.
2. $\vdash_{\mathscr{L}} (\varphi_S)^\sharp \leftrightarrow \varphi$.

Proof. Claim 1: For all $\psi \in \mathsf{Prop}(\Lambda(\mathscr{P}(S)))$, $\vdash^1_{\mathscr{L}} \psi$ implies that $\vdash_{\mathscr{L}} \psi^\sharp$.
It is clear that Item 1 follows from Claim 1 - let us now prove Claim 1: Suppose that $\vdash^1_{\mathscr{L}} \psi$, ie., assume that ψ is one-step $\mathscr{L}$-derivable. By the definition of one-step derivability, this means that the set $\{\chi\sigma \mid \chi \in \mathrm{Ax}, \sigma : P \to \mathscr{P}(S)\}$ propositionally entails ψ. This implies that $\psi^\sharp$ is a propositional consequence of the set $W = \{\chi\sigma^\sharp \mid \chi \in \mathrm{Ax}, \sigma : P \to \mathscr{P}(S)\}$. Any formula $\chi\sigma^\sharp \in W$ can be written as $\chi\tau$ with $\tau : P \to \mathsf{Prop}(\Phi)$ defined as $\tau(p) = \xi_{\sigma(p)}$ - in other words, all elements of W are substitution instances of $\mathscr{L}$-axioms, $\psi^\sharp$ is a propositional consequence of W and hence, as $\mathscr{L}$ is closed under propositional reasoning and uniform substitution, we get $\vdash_{\mathscr{L}} \psi^\sharp$ as required.

It remains to prove item 2. We prove that for all $\varphi \in \mathsf{Prop}(\Phi)$,

$$\vdash_{\mathscr{L}} \varphi \leftrightarrow (\varphi_S)^\sharp \tag{2}$$

Item 2 then follows by applying the congruence rule and propositional logic. For (2), it is easy to see that for all $\varphi \in \mathsf{Prop}(\Phi)$, $\vdash_{PL} (\varphi_S)^\sharp \to \varphi$ and hence $\vdash_{\mathscr{L}} (\varphi_S)^\sharp \to \varphi$. For the other implication, suppose towards a contradiction that $\varphi \wedge \neg(\varphi_S)^\sharp$ is $\mathscr{L}$-consistent. Then there is a maximally $\mathscr{L}$-consistent set Ξ such that $\varphi, \neg(\varphi_S)^\sharp \in \Xi$. Take $\Delta := \Xi \cap \Phi$. We have

$$\text{for all } \psi \in \mathsf{Prop}(\Phi): \quad \Delta \vdash_{PL} \psi \quad \text{or} \quad \Delta \vdash_{PL} \neg\psi \tag{3}$$

The proof is by induction on ψ. The base case where $\psi \in \Phi$ is trivial. If $\psi = \neg\psi'$, then by I.H. $\Delta \vdash_{PL} \psi'$ or $\Delta \vdash_{PL} \neg\psi'$ and it follows that $\Delta \vdash_{PL} \neg\psi$ or $\Delta \vdash_{PL} \psi$. If $\psi = \psi_1 \wedge \psi_2$, then by I.H. we have:

$$(\Delta \vdash_{PL} \psi_1 \quad \text{or} \quad \Delta \vdash_{PL} \neg\psi_1) \qquad \text{and} \qquad (\Delta \vdash_{PL} \psi_2 \quad \text{or} \quad \Delta \vdash_{PL} \neg\psi_2).$$

Considering all four combinations yields $\Delta \vdash_{PL} \psi_1 \wedge \psi_2$ or $\Delta \vdash_{PL} \neg(\psi_1 \wedge \psi_2)$.

From (3) and $\varphi \in \Xi$, we obtain that $\Delta \vdash_{PL} \varphi$. On the other hand, from $\neg(\varphi_S)^\sharp \in \Xi$ it follows that $\Delta \nvdash_{PL} (\varphi_S)^\sharp$, and hence, because $(\varphi_S)^\sharp = \bigvee\{\bigwedge\Delta \mid \Delta \in S, \Delta \vdash_{PL} \varphi\}$, we have $\Delta \nvdash_{PL} \varphi$. Thus we have a contradiction, and we conclude that $\varphi \wedge \neg(\varphi_S)^\sharp$ is $\mathscr{L}$-inconsistent which proves that $\vdash_{\mathscr{L}} \varphi \to (\varphi_S)^\sharp$. QED

Lemma 3.8 (Existence lemma) *Assume that $\mathscr{L}_\Diamond$ is one-step complete for T. For all $\alpha \in A$ and all $\Gamma \in S$ there is a $t_{\alpha,\Gamma} \in T(S)$ such that for all $U \subseteq S$,*

1. *If $\Gamma \vdash_{\mathscr{L}} \langle\alpha\rangle\xi_U$ then $t_{\alpha,\Gamma} \in \lambda_S(U)$.*
2. *If $\Gamma \vdash_{\mathscr{L}} \neg\langle\alpha\rangle\xi_U$ then $t_{\alpha,\Gamma} \in \lambda_S(U)$.*
3. *If $\Gamma \nvdash_{\mathscr{L}} \langle\alpha\rangle\xi_U$ and $\langle\alpha\rangle\xi_U \wedge \Gamma$ is $\mathscr{L}$-consistent, then $t_{\alpha,\Gamma} \in \lambda_S(U)$.*

It follows that for all $\alpha \in A$ and all $\Gamma \in S$ there is a $t_{\alpha,\Gamma} \in T(S)$ such that for all $U \subseteq S$,

$$t_{\alpha,\Gamma} \in \lambda_S(U) \quad \textit{iff} \quad \Gamma \wedge \langle\alpha\rangle\xi_U \textit{ is } \mathscr{L}\textit{-consistent}. \tag{4}$$

Proof. We spell out the details of the proof for the case that λ is a diamond-like lifting. For the case that λ is box-like the roles of the positive and negative formulas of the form $\langle\alpha\rangle\varphi$ and $\neg\langle\alpha\rangle\varphi$ in the proof have to be switched. We now turn to the proof of the lemma.

Suppose for a contradiction that there is $\alpha \in A$ and $\Gamma \in S$ such that no $t \in TS$ satisfies conditions 1 and 2 of the lemma. Consider the formula

$$\varphi(\Gamma) = \bigvee\{\langle\alpha\rangle\xi_X \mid X \subseteq S, \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\} \vee \bigvee\{\neg\langle\alpha\rangle\xi_X \mid X \subseteq S, \Gamma \vdash_{PL} \langle\alpha\rangle\xi_X\}$$

and note that

$$\varphi(\Gamma)_S = \bigvee\{\langle\alpha\rangle X \mid X \subseteq S, \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\} \vee \bigvee\{\neg\langle\alpha\rangle X \mid X \subseteq S, \Gamma \vdash_{PL} \langle\alpha\rangle\xi_X\}$$

Then by our assumption on α and Γ we have $[\![\varphi(\Gamma)_S]\!]_1 = (TS)^A$. Recall from Lemma 3.2 that one-step completeness of $\mathscr{L}_\Diamond$ implies one-step completeness of $\mathscr{L}$ wrt T^A. Therefore we obtain that $\vdash^1_{\mathscr{L}} \varphi(\Gamma)_S$ and thus, by Lemma 3.7, that $\vdash_{\mathscr{L}} \varphi(\Gamma)$. This yields a contradiction with our assumption that Γ is $\mathscr{L}$-consistent. For each $\Gamma \in S$ and $\alpha \in A$ we fix an element $s_{\alpha,\Gamma} \in TS$ satisfying conditions 1 and 2.

Consider now $\Gamma \in S$ and let $U \subseteq S$ be such that $\Gamma \nvdash_{\mathscr{L}} \langle\alpha\rangle\xi_U$ and $\langle\alpha\rangle\xi_U \wedge \Gamma$ is $\mathscr{L}$-consistent. As $\langle\alpha\rangle\xi_U \wedge \Gamma$ is $\mathscr{L}$-consistent the set $\{\langle\alpha\rangle\xi_U\} \cup \{\neg\langle\alpha\rangle\xi_X \mid \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\}$ is $\mathscr{L}$-consistent and we can easily show - using Lemma 3.7 - that the set $\{\langle\alpha\rangle U\} \cup \{\neg\langle\alpha\rangle X \mid \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\}$ is one-step $\mathscr{L}$-consistent. Therefore by one-step completeness of $\mathscr{L}$ there must be an $f_{\Gamma,U} \in (TS)^A$ such that

$$f_{\Gamma,U} \models^1 \bigwedge(\{\langle\alpha\rangle U\} \cup \{\neg\langle\alpha\rangle X \mid \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\})$$

or, equivalently,

$$f_{\Gamma,U}(\alpha) \in \bigcap(\{\lambda_S(U)\} \cup \{S \setminus \lambda_S(X) \mid \Gamma \vdash_{PL} \neg\langle\alpha\rangle\xi_X\}).$$

Using the fact that λ is diamond-like we can now easily verify that for each $\Gamma \in S$ and $\alpha \in A$ the join $t_{\alpha,\Gamma} := \bigvee_{U \in \Xi} f_{\Gamma,U}(\alpha) \vee s_{\alpha,\Gamma}$ with $\Xi = \{U \subseteq X \mid \Gamma \nvdash_{\mathscr{L}} \langle\alpha\rangle\xi_U \text{ and } \langle\alpha\rangle\xi_U \wedge \Gamma \text{ is } \mathscr{L}\text{-consistent}\}$ satisfies all conditions of the lemma. QED

Proposition 3.6 now follows immediately from Lemma 3.8 by taking $\widehat{\gamma}(\alpha)(\Gamma) := t_{\alpha,\Gamma}$ for all $\alpha \in A_0$.

3.2 Standard, coherent models

We saw in the previous subsection that one-step completeness ensures the existence of a strongly coherent structure. However, this structure is not necessarily standard. We now show that from a strongly coherent structure, we can obtain a standard model which satisfies the usual coherence condition by extending the strongly structure inductively from atomic actions to all actions $\alpha \in A$ and proving that the resulting structure map $\gamma\colon S \to (TS)^A$ is coherent.

We start by defining a $\gamma\colon S \to (TS)^A$ which is almost standard. For technical reasons, we define γ on tests from Φ in terms of membership. Once we prove that truth is membership (Lemma 3.16), it follows that γ is standard. This way we avoid a mutual induction argument.

Definition 3.9 (Coherent dynamic structure) Let $\gamma_0\colon S \to (TS)^A$ be the strongly coherent structure that exists by Proposition 3.6. Define $\gamma\colon S \to (TS)^A$ inductively as follows:

$$\begin{array}{rcl}
\widehat{\gamma}(\alpha) & := & \widehat{\gamma}_0(\alpha) \qquad \text{for } \alpha \in A_0 \\
\widehat{\gamma}(\varphi?)(\Gamma) & := & \begin{cases} \eta_S(\Gamma) & \text{if } \varphi \in \Gamma \quad \text{and} \quad \varphi \in \Phi \\ \eta_S(\Gamma) & \text{if } \Gamma \in [\![\varphi]\!]_{(X,\gamma,V)} \quad \text{and} \quad \varphi \notin \Phi \\ \bot_{TS} & \text{otherwise.} \end{cases} \\
\widehat{\gamma}(\sigma(\alpha_1,\ldots,\alpha_n))(\Gamma) & := & \sigma_S(\widehat{\gamma}(\alpha_1)(\Gamma),\ldots,\widehat{\gamma}(\alpha_n)(\Gamma))) \\
\widehat{\gamma}(\alpha^*)(\Gamma) & := & \widehat{\gamma}(\alpha)^*(\Gamma)
\end{array}$$

where V is the canonical valuation $V(p) = \{\Delta \in S \mid p \in \Delta\}$. ◁

The rest of the section will be dedicated to proving that γ is in fact coherent. This can be done largely similarly to what we did in our previous work [6] for the iteration-free case. The main difference is obviously the presence of the $*$-operator. Here a crucial role is played by the following monotone operator on $\mathscr{P}(S)$ that allows us to formalise a logic-induced notion of reachability.

Definition 3.10 (F^X_β) For $\beta \in A$ and $X \subseteq S$ we define an operator

$$\begin{array}{rcl}
F^X_\beta : \mathscr{P}S & \to & \mathscr{P}S \\
Y & \mapsto & \{\Delta \in S \mid \Delta \wedge \langle\beta\rangle\xi_Y \text{ consistent}\} \cup X
\end{array}$$

It is easy to see that this is a monotone operator, its least fixpoint will be denoted by Z^X_β. ◁

Lemma 3.11 *For all $\Delta \in S$ and all $X \subseteq S$ we have: $\Delta \wedge \langle\beta\rangle\xi_{Z^X_\beta}$ is consistent $\quad\Rightarrow\quad$ $\Delta \in Z^X_\beta$.*

Proof. This is an immediate consequence of the fact that Z^X_β is a fixpoint of F^X_β. QED

The following technical lemma is required for the inductive proof of the first coherence Lemma 3.14.

Lemma 3.12 *Let $\beta \in A$ be an action such that for all $\Gamma \in S$ and all $X \subseteq S$ we have*

$$\Gamma \wedge \langle\beta\rangle\xi_X \text{ consistent} \quad\Rightarrow\quad \widehat{\gamma}(\Gamma) \in \lambda_S(X).$$

Then $\Gamma \in Z^X_\beta$ implies $\widehat{\gamma}(\beta^*)(\Gamma) \in \lambda_S(X)$.

Proof. This proof is using our assumption that λ is diamond-like. Recall first that by definition we have $\widehat{\gamma}(\beta^*) = \widehat{\gamma}(\beta)^*$, thus we need to show that $\widehat{\gamma}(\beta)^*(\Gamma) \in \lambda_S(X)$. Let $Y = \{\Delta \in S \mid \widehat{\gamma}(\beta)^*(\Delta) \in \lambda_S(X)\}$. In order to prove our claim it suffices to show that $F_\beta^X(Y) \subseteq Y$, ie, that Y is a prefixed point of F_β^X (as Z_β^X is the smallest such prefixed point and as $Z_\beta^X \subseteq Y$ is equivalent to the claim of the lemma). Let $\Gamma \in F_\beta^X(Y)$. We need to show that $\Gamma \in Y$. In case $\Gamma \in X$ we have $\widehat{\gamma}^0(\Gamma) = \eta(\Gamma) \in \lambda_S(\hat{\varphi})$ because $\eta(\Gamma) \in \lambda_S(\hat{\varphi})$ is equivalent to $\Gamma \in X$ as $\widehat{\lambda}$ is a monad morphism. Suppose now that $\Gamma \wedge \langle\beta\rangle\xi_Y$ is consistent. By our assumption on β this implies that

$$\widehat{\gamma}(\beta)(\Gamma) \in \lambda_S(Y) = \lambda_S(\{\Delta \mid \widehat{\gamma}(\beta)^*(\Delta) \in \lambda_S(X)\}).$$

Using Lemma 2.8 this implies

$$(\widehat{\gamma}(\beta) * \widehat{\gamma}(\beta)^*)(\Gamma) \in \lambda_S(X)$$

and

$$\widehat{\gamma}(\beta) * \widehat{\gamma}(\beta)^*(\Gamma) = (\widehat{\gamma}(\beta) * \bigvee_i \widehat{\gamma}(\beta)^{[i]})(\Gamma) = \bigvee_i \widehat{\gamma}(\beta)^{[i+1]}(\Gamma)$$

where the last equality follows from the fact that we are working with a monad T whose Kleisli composition left-distributes over joins. As λ is assumed to be diamond-like, it follows that there is a $j \geq 1$ such that $\widehat{\gamma}(\beta)^{[j]}(\Gamma) \in \lambda_S(X)$ and thus $\Gamma \in Y$ as required. QED

We are now ready to prove two crucial coherence lemmas. As we are ultimately only interested in the truth of formulas in Φ we can confine ourselves to what we call *relevant* actions:

Definition 3.13 (Relevant test, relevant action) A test $\varphi?$ is called *relevant* if $\varphi \in \Phi$. An action $\alpha \in A$ is called *relevant* if it only contains relevant tests. ◁

The following lemma proves the first half of the announced coherence.

Lemma 3.14 *For all relevant actions $\alpha \in A$, $\Gamma \in S$ and all $X \subseteq S$ we have*

$$\Gamma \wedge \langle\alpha\rangle\xi_X \text{ consistent} \quad \Rightarrow \quad \widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(X).$$

Proof. By induction on α. The base case holds trivially as γ is strongly coherent for all atomic actions. Let $\alpha = \varphi?$ for some $\varphi \in \Phi$ (here we can assume $\varphi \in \Phi$ as we only consider relevant actions) and suppose $\Gamma \wedge \langle\varphi?\rangle\xi_X$ is consistent for some $X \subseteq S$. Then, as λ is diamond-like, we have $\Gamma \wedge \varphi \wedge \xi_X$ is consistent. This implies $\varphi \in \Gamma$ and $\Gamma \in X$. As $\varphi \in \Gamma$, we have by the definition of γ that $\widehat{\gamma}(\varphi?)(\Gamma) = \eta_S(\Gamma)$ and thus $\Gamma \in X$ implies $\widehat{\gamma}(\varphi?)(\Gamma) \in \lambda_S(X)$ as required.

For an n-ary pointwise operation $\sigma \in \Sigma$, we want to show that

$$\Gamma \wedge \langle\underline{\sigma}(\alpha_a, \ldots, \alpha_n)\rangle\xi_X \text{ consistent} \quad \Rightarrow \quad \sigma_S^S(\widehat{\gamma}(\alpha_1)(\Gamma), \ldots, \widehat{\gamma}(\alpha_n)(\Gamma)) \in \lambda_S(X)$$

Using the σ-axiom and that $\widehat{\lambda} \circ \sigma = \chi \circ \widehat{\lambda}^n$, this is equivalent to

$$\Gamma \wedge \varphi(\check{\chi}, \alpha_1, \ldots, \alpha_n, \xi_X) \text{ consistent} \quad \Rightarrow \quad X \in \chi_S(\widehat{\lambda}(\widehat{\gamma}(\alpha_1)(\Gamma)), \ldots, \widehat{\lambda}(\widehat{\gamma}(\alpha_n)(\Gamma))) \tag{5}$$

and (5) can be proved by induction on $\check{\chi}$ in a manner very similar to the one used in the proof of Lemma 27 in [6].

Suppose α is of the form $\alpha = \beta_0;\beta_1$ and suppose $\Gamma \wedge \langle\beta_0;\beta_1\rangle\xi_U$ is consistent for some $U \subseteq S$. Using the compositionality axiom we have $\vdash_{\mathscr{L}} \langle\beta_0;\beta_1\rangle\xi_U \leftrightarrow \langle\beta_0\rangle\langle\beta_1\rangle\xi_U$. Therefore $\Gamma \wedge \langle\beta_0\rangle\langle\beta_1\rangle\xi_U$ is

consistent. This implies in turn that $\Gamma \wedge \langle\beta_0\rangle(\top \wedge \langle\beta_1\rangle\xi_U)$ is consistent and, as $\vdash_{\mathscr{L}} \top \leftrightarrow \bigvee_{\Delta\in S}\bigwedge\Delta$ by Lemma 3.7, we obtain that $\Gamma \wedge \langle\beta_0\rangle((\bigvee_{\Delta\in S}\bigwedge\Delta) \wedge \langle\beta_1\rangle\xi_U)$ and thus $\Gamma \wedge \langle\beta_0\rangle(\bigvee_{\Delta\in S}\bigwedge(\Delta \wedge \langle\beta_1\rangle\xi_U))$ is consistent. Clearly the latter implies that $\Gamma \wedge \langle\beta_0\rangle(\bigvee_{\Delta\in Y}\bigwedge(\Delta \wedge \langle\beta_1\rangle\xi_U))$ is consistent for $Y := \{\Delta \in S \mid \Delta \wedge \langle\beta_1\rangle\xi_U \text{ consistent}\}$. Therefore we also have $\Gamma \wedge \langle\beta_0\rangle\xi_Y$ is consistent. Now we apply the induction hypothesis to get

$$\widehat{\gamma}(\beta_0)(\Gamma) \in \lambda_S(Y) = \lambda_S(\{\Delta \in S \mid \Delta \wedge \langle\beta_1\rangle\xi_U \text{ consistent}\}) \overset{\text{I.H.}}{\subseteq} \lambda_S(\{\Delta \in S \mid \widehat{\gamma}(\beta_1)(\Delta) \in \lambda_S(U)\})$$

and by Lemma 2.8 we conclude that $\widehat{\gamma}(\beta_0;\beta_1)(\Gamma) = \widehat{\gamma}(\beta_0) * \widehat{\gamma}(\beta_1)(\Gamma) \in \lambda_S(U)$.

Suppose now $\alpha = \beta^*$. It follows from Lemma 3.12 and the I.H. on β that $\Gamma \in Z^X_\beta$ implies $\widehat{\gamma}(\beta^*)(\Gamma) \in \lambda_S(X)$. Therefore it suffices to prove that $\Gamma \wedge \langle\beta^*\rangle\xi_X$ is consistent implies $\Gamma \in Z^X_\beta$.

Suppose that $\Gamma \wedge \langle\beta^*\rangle\xi_X$ is consistent and recall the $\diamond$-induction rule:

$$\frac{\vdash \langle\beta\rangle\psi \vee \varphi \to \psi}{\vdash \langle\beta^*\rangle\varphi \to \psi}$$

Our claim is that

$$\vdash \langle\beta\rangle\xi_{Z^X_\beta} \vee \xi_X \to \xi_{Z^X_\beta} \tag{+}$$

Before we prove (+) let us see why it suffices to complete the proof: If (+) holds, we can apply the induction rule in order to obtain

$$\vdash \langle\beta^*\rangle\xi_X \to \xi_{Z^X_\beta}. \tag{6}$$

By assumption we have $\Gamma \wedge \langle\beta^*\rangle\xi_X$. Together with (6) this implies that $\Gamma \wedge \xi_{Z^X_\beta}$ are consistent and thus, by Lemma 3.11, that $\Gamma \in Z^X_\beta$ as required.

Proof of (+): Suppose for a contradiction that (+) does not hold. This implies that $(\langle\beta\rangle\xi_{Z^X_\beta} \vee \xi_X) \wedge \neg\xi_{Z^X_\beta}$ is consistent. We distinguish two cases.

Case 1 $\langle\beta\rangle\xi_{Z^X_\beta} \wedge \neg\xi_{Z^X_\beta}$ is consistent. Then there is a maximal consistent set Ξ such that $\langle\beta\rangle\xi_{Z^X_\beta}, \neg\xi_{Z^X_\beta} \in \Xi$. Let $\Delta := \Xi \cap \Phi$. By definition and (3) we know that $\Delta \vdash_{\mathscr{L}} \neg\xi_{Z^X_\beta}$ and thus $\Delta \in S \setminus Z^X_\beta$. Furthermore $\Delta \wedge \langle\beta\rangle\xi_{Z^X_\beta}$ is consistent. The latter implies, again by Lemma 3.11, that $\Delta \in Z^X_\beta$ which is a contradiction and we conclude that $\langle\beta\rangle\xi_{Z^X_\beta} \wedge \neg\xi_{Z^X_\beta}$ cannot be consistent.

Case 2 $\xi_X \wedge \neg\xi_{Z^X_\beta}$ is consistent. Again - using a similar argument to the previous case - this implies that there is an atom $\Delta \in S \setminus Z^X_\beta$ such that $\Delta \wedge \xi_X$ is consistent. But the latter entails that $\Delta \in X \subseteq Z^X_\beta$ which yields an obvious contradiction. QED

Lemma 3.15 *For all $\langle\alpha\rangle\varphi \in \Phi$ and all $\Gamma \in S$ we have*

$$\widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(\hat{\varphi}) \quad \Rightarrow \quad \langle\alpha\rangle\varphi \in \Gamma.$$

Proof. Again this is proven by induction on α. Let $\alpha = \psi?$ and suppose $\widehat{\gamma}(\psi?)(\Gamma) \in \lambda_S(\hat{\varphi})$ for some $\langle\psi?\rangle\varphi \in \Phi$. As λ is diamond-like, we have $\widehat{\gamma}(\psi?)(\Gamma) \neq \bot$ and thus, by the definition of $\widehat{\gamma}$, we have $\psi \in \Gamma$ and $\eta_S(\Gamma) \in \lambda_S(\hat{\varphi})$. The latter implies $\Gamma \in \hat{\varphi}$, ie, $\varphi \in \Gamma$. Both $\psi \in \Gamma$ and $\varphi \in \Gamma$ imply, using the axiom $\vdash_{\mathscr{L}} \langle\psi?\rangle\varphi \leftrightarrow \psi \wedge \varphi$, that $\langle\psi?\rangle\varphi \in \Gamma$ as required.

Let α be of the form $\alpha = \beta^*$ and let $\Gamma \in S$ be such that $\widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(\hat{\varphi})$. Then $\widehat{\gamma}(\alpha) = \widehat{\gamma}(\beta)^*$ and thus we have $\widehat{\gamma}(\beta)^*(\Gamma) \in \lambda_S(\hat{\varphi})$. This means that $\bigvee_j \widehat{\gamma}(\beta)^{[j]}(\Gamma) \in \lambda_S(\hat{\varphi})$. By diamond-likeness of λ this is equivalent to the existence of one $j \geq 0$ such that $\widehat{\gamma}(\beta)^{[j]}(\Gamma) \in \lambda_S(\hat{\varphi})$.

In case $j = 0$ we can easily see that $\Gamma \in \hat{\varphi}$, ie, $\varphi \in \Gamma$ which implies - using the axiom $(\langle\beta\rangle\langle\beta^*\rangle\varphi \vee \varphi) \leftrightarrow \langle\beta^*\rangle\varphi$ - that $\langle\beta^*\rangle\varphi \in \Gamma$.

Suppose now $j = m+1$, ie, $\widehat{\gamma}(\beta)^{[m+1]}(\Gamma) \in \lambda_S(\hat{\varphi})$. By Lemma 2.8 this implies that

$$\widehat{\gamma}(\beta)(\Gamma) \in \lambda_S\left(\{\Delta \mid \widehat{\gamma}(\beta)^{[m]}(\Delta) \in \lambda_S(\hat{\varphi})\}\right).$$

By I.H. on m we have $\{\Delta \mid \widehat{\gamma}(\beta)^{[m]}(\Delta) \in \lambda(\hat{\varphi})\} \subseteq \widehat{\langle\beta^*\rangle\varphi}$ and hence, by monotonicity of λ, that

$$\widehat{\gamma}(\beta)(\Gamma) \in \lambda_S(\widehat{\langle\beta^*\rangle\varphi}).$$

By I.H. on β this implies that $\langle\beta\rangle\langle\beta^*\rangle\varphi \in \Gamma$ and thus - using again the same axiom as in the base case - that $\langle\beta^*\rangle\varphi \in \Gamma$. QED

Lemma 3.16 (Dynamic truth lemma) *The coalgebra structure $\gamma : S \to (TS)^A$ from Def. 3.9 together with the valuation $V : P \to \mathscr{P}(S)$ given by $V(p) = \hat{p}$ for $p \in P_0$ forms a θ-dynamic $\mathbb{T}$-model such that for all $\varphi \in \Phi$ we have $[\![\varphi]\!] = \hat{\varphi}$.*

Proof. It follows from Lemma 3.14 and Lemma 3.15 that for all $\langle\alpha\rangle\varphi \in \Phi$ we have

$$\langle\alpha\rangle\varphi \in \Gamma \quad \text{iff} \quad \widehat{\gamma}(\alpha)(\Gamma) \in \lambda_S(\hat{\varphi}).$$

Therefore it follows by Lemma 3.4 that $[\![\varphi]\!] = \hat{\varphi}$ for all $\varphi \in \Phi$ as required. In particular this shows that the resulting model is θ-dynamic, since for all relevant tests $\varphi?$ we have $\varphi \in \Gamma$ iff $\Gamma \in [\![\varphi]\!]$. QED

Theorem 3.17 *If $\mathscr{L}_\Diamond = (\{\Diamond\}, \mathrm{Ax}, \emptyset, \mathrm{Ru})$ is one-step complete with respect to the T-coalgebraic semantics given by λ, and θ consists of positive operations, then the dynamic logic $\mathscr{L} = \mathscr{L}(\theta, ;, ^*, ?)$ is (weakly) complete with respect to the class of all θ-dynamic $\mathbb{T}$-models.*

Proof. Assume that ψ is an $\mathscr{L}$-consistent formula. Let S be the set of $\mathscr{L}$-atoms over $\Phi = Cl(\psi)$ and let γ: $S \to (TS)^A$ be defined as in Definition 3.9 and V the valuation given by $V(p) = \hat{p}$ for $p \in P_0$. By Lemma 3.16, $\mathbb{M} = (S, \gamma, \lambda, V)$ is a θ-dynamic $\mathbb{T}$-model. Since ψ is $\mathscr{L}$-consistent there is an $\mathscr{L}$-atom $\Delta \in S$ that contains ψ and hence by the Dynamic Truth Lemma 3.16, ψ is true at Δ in $\mathbb{M}$. QED

As corollaries to our main theorem we obtain completeness for a number of concrete dynamic modal logics.

Corollary 3.18 *(i) We recover the classic result that PDL is complete with respect to $\cup$-dynamic $\mathscr{P}$-models from the fact that the diamond version of the modal logic **K** is one-step complete with respect to $\mathscr{P}$ (cf. [17]), $\cup$ is a positive natural operation on $\mathscr{P}$, and the Kripke diamond $\lambda_X(U) = \{V \in \mathscr{P}X \mid V \cap U \neq \emptyset\}$ is monotonic and its transpose is a monad morphism. (ii) Taking as base logic $\mathscr{L}_\Diamond$ the monotonic modal logic* **M** *with semantics given by the usual monotonic neighbourhood predicate lifting $\lambda_X(U) = \{N \in \mathscr{M}X \mid U \in N\}$ with rank-1 axiomatisation $\mathrm{Ax} = \{\Diamond(p \wedge q) \to \Diamond p\}$, it is well known that $\mathscr{L}_\Diamond$ is one-step complete for $\mathscr{M}$, see also [6]. Since $\cup$ is a positive natural operation on $\mathscr{M}$, we get that dual-free GL is complete with respect to $\cup$-dynamic $\mathscr{M}$-models. (iii) Similarly, dual-free GL with intersection is complete with respect to $\cup, \cap$-dynamic $\mathscr{M}$-models.*

4 Conclusion

There are several ways in which to continue our research. Firstly we will look for other, new examples that fit into our general coalgebraic framework. A first good candidate seems to be the filter monad $\mathscr{F}$ (cf. [5, 8, 20]). It is easy to see that taking upsets yields a monad morphism $\tau\colon \mathscr{P} \Rightarrow \mathscr{F}$ and the induced join on $\mathscr{F}X$ is intersection of filters. We note that filters are not closed under unions (only under updirected unions), so $\cup$ is not a natural operation on $\mathscr{F}$. Taking $\mathscr{L}_\Diamond$ to be the diamond version of modal logic **K**, and $\lambda : \mathscr{Q} \Rightarrow \mathscr{Q} \circ \mathscr{F}$ to be $\lambda_X(U) = \{F \in \mathscr{F}X \mid X \setminus U \notin F\}$ (i.e., the dual of the usual neigbourhood modality), then $\mathscr{L}_\Diamond$ is complete with respect to the class of all $\mathscr{F}$-coalgebras, since any Kripke model $(X, \rho\colon X \to \mathscr{P}X, V)$ is pointwise equivalent with the $\mathscr{F}$-model $(X, \tau \circ \rho\colon X \to \mathscr{F}X, V)$, hence any φ that can be falsified in a Kripke model can also be falsified in a filter coalgebra, cf. [2]. We conjecture that $\mathscr{L}_\Diamond$ is one-step complete for $\mathscr{F}$ and λ. From this, a completeness result would follow for a new PDL-like logic for the filter monad with intersection on actions.

Secondly, we will study variations of our coalgebraic framework to monads that carry quantitative information to cover important cases such as probabilistic and weighted transition systems. We expect that we need to switch to a multivalued logic, using for example $T(1)$ as truth value object, as in [3]. In general, we would also like to better understand how our exogenous logics relate to the endogenous coalgebraic logics of [3] and the weakest preconditions arising from state-and-effect triangles in, e.g., [8, 9]. One difference is that in [3], the monad T is assumed to be commutative. This condition ensures that the Kleisli category is enriched over Eilenberg-Moore algebras. This could be an interesting approach to obtaining a "canonical" algebra of program operations, even though, Eilenberg-Moore algebras do not have canonical representations in terms of operations and equations. Moreover, one of our main example monads, the monotonic neighbourhood monad is not commutative, but it is still amenable to our framework.

Finally, our most ambitious aim will be to extend our coalgebraic framework to a completeness proof which will entail completeness of full GL which remains an open problem [15]. One reason that this is a difficult problem is that, unlike PDL, full GL is able to express fixpoints of arbitrary alternation depth [1].

References

[1] D. Berwanger (2003): *Game Logic is strong enough for parity games*. *Studia Logica* 75(2), pp. 205–219, doi:10.1023/A:1027358927272.

[2] B. F. Chellas (1980): *Modal Logic - An Introduction*. Cambridge University Press, doi:10.1017/CBO9780511621192.

[3] C. Cîrstea (2014): *A Coalgebraic Approach to Linear-Time Logics*. In A. Muscholl, editor: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Proceedings*, *LNCS* 8412, Springer, pp. 426–440, doi:10.1007/978-3-642-54830-7_28.

[4] M. J. Fischer & R. F. Ladner (1979): *Propositional dynamic logic of regular programs*. *J. of Computer and System Sciences* 18, pp. 194–211, doi:10.1016/0022-0000(79)90046-1.

[5] H. Peter Gumm (2005): *From* T*-Coalgebras to Filter Structures and Transition Systems*. In: *Algebra and Coalgebra in Computer Science: First International Conference, CALCO 2005, Swansea, UK, September 3-6, 2005, Proceedings*, *LNCS* 3629, Springer, pp. 194–212, doi:10.1007/11548133_13.

[6] H.H. Hansen, C. Kupke & R.A. Leal (2014): *Strong Completeness for Iteration-Free Coalgebraic Dynamic Logics*. Technical Report, ICIS, Radboud University Nijmegen. Available at `https://pms.cs.ru.nl/iris-diglib/src/icis_tech_reports.php`. See also updated version at `http://homepage.tudelft.nl/c9d1n/papers/cpdl-techrep.pdf`.

[7] H.H. Hansen, C. Kupke & R.A. Leal (2014): *Strong completeness of iteration-free coalgebraic dynamic logics*. In J. Diaz, I. Lanese & D. Sangiorgi, editors: *Theoretical Computer Science (TCS 2014). 8th IFIP TC 1/WG 2.2 International Conference, LNCS* 8705, Springer, pp. 281–295, doi:10.1007/978-3-662-44602-7_22.

[8] B. Jacobs (2015): *A recipe for state-and-effect triangles*. In: *Algebra and Coalgebra in Computer Science: Sixth International Conference (CALCO 2015), Proceedings*, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, doi:10.4230/LIPIcs.CALCO.2015.113.

[9] Bart Jacobs (2015): *Dijkstra and Hoare monads in monadic computation*. *Theoretical Computer Science*, doi:10.1016/j.tcs.2015.03.020. Article in Press.

[10] D. Kozen (1983): *Results on the propositional mu-calculus*. *Theoretical Computer Science* 27, pp. 333–354, doi:10.1016/0304-3975(82)90125-6.

[11] D. Kozen & R. Parikh (1981): *An elementary proof of the completeness of PDL*. *Theoretical Computer Science* 14, pp. 113–118, doi:10.1016/0304-3975(81)90019-0.

[12] C. Kupke & D. Pattinson (2011): *Coalgebraic semantics of modal logics: an overview*. *Theoretical Computer Science* 412(38), pp. 5070–5094, doi:10.1016/j.tcs.2011.04.023.

[13] S. MacLane (1998): *Categories for the Working Mathematician*, 2nd edition. Springer.

[14] R. Parikh (1985): *The logic of games and its applications*. In: *Topics in the Theory of Computation, Annals of Discrete Mathematics* 14, Elsevier, doi:10.1016/S0304-0208(08)73078-0.

[15] M. Pauly & R. Parikh (2003): *Game Logic: An Overview*. *Studia Logica* 75(2), pp. 165–182, doi:10.1023/A:1027354826364.

[16] J. J. M. M. Rutten (2000): *Universal Coalgebra: A Theory of Systems*. *Theoretical Computer Science* 249, pp. 3–80, doi:10.1016/S0304-3975(00)00056-6.

[17] L. Schröder & D. Pattinson (2009): *Strong completeness of coalgebraic modal logics*. In: *Proceedings of STACS 2009*, pp. 673–684, doi:10.4230/LIPIcs.STACS.2009.1855.

[18] Lutz Schröder (2007): *A finite model construction for coalgebraic modal logic*. *J. Log. Algebr. Program.* 73(1-2), pp. 97–110, doi:10.1016/j.jlap.2006.11.004.

[19] I. Walukiewicz (2000): *Completeness of Kozen's Axiomatisation of the Propositional μ-Calculus*. *Inf. Comput.* 157(1-2), pp. 142–182, doi:10.1006/inco.1999.2836.

[20] O. Wyler (1981): *Algebraic theories of continuous lattices*. In B. Banaschewski & R.-E. Hoffman, editors: *Continuous Lattices, Lect. Notes Math.* 871, Springer, Berlin, pp. 187–201, doi:10.1007/978-3-642-61598-6_11.

The Arity Hierarchy in the Polyadic μ-Calculus

Martin Lange
School of Electrical Engineering and Computer Science, University of Kassel, Germany

The polyadic μ-calculus is a modal fixpoint logic whose formulas define relations of nodes rather than just sets in labelled transition systems. It can express exactly the polynomial-time computable and bisimulation-invariant queries on finite graphs. In this paper we show a hierarchy result with respect to expressive power inside the polyadic μ-calculus: for every level of fixpoint alternation, greater arity of relations gives rise to higher expressive power. The proof uses a diagonalisation argument.

1 Introduction

The modal μ-calculus $\mathcal{L}_\mu$ is a well-studied logic [14, 4, 5], obtained by adding restricted second-order quantification in the form of least and greatest fixpoints to a multi-modal logic interpreted over labelled transition systems. A formula of the modal μ-calculus is thus interpreted in a state of such transition systems which means that such formulas *define* sets of states in transition systems. For example, $\nu X.\mu Y.\langle a\rangle X \vee \langle b\rangle Y$ defines the set of all states from which there is a path with labels 'a' and 'b' that contains infinitely many occurrences of the symbol 'a'.

The polyadic μ-calculus $\mathcal{L}_\mu^\omega$ is a much less known extension of the modal μ-calculus whose formulas define *relations* rather than sets of states. They are interpreted in a tuple of states rather than a single state, and there are modal operators for each position in this tuple. Thus, one states "the third state has an 'a'-successors" for instance rather than just "there is an 'a'-successors." Combining such simple modal statements with fixpoint quantifiers yields an expressive logic with interesting applications: the polyadic μ-calculus was first defined by Andersen [2] and used as a logic for defining process equivalences like bisimilarity [15, 16]. Later it was re-invented by Otto under the name *Higher-Dimensional μ-Calculus* [19] and shown to capture the complexity class P over bisimulation-invariant class of finite graphs. I.e. a bisimulation-invariant property of finite graphs can be computed in polynomial time iff it is definable in $\mathcal{L}_\mu^\omega$.

There is a natural hierarchy in $\mathcal{L}_\mu^\omega$ given by fragments of bounded arity. The polyadic μ-calculus itself can be seen as a fragment of FO+LFP, i.e. First-Order Logic extended with fixpoint quantifiers. The translation naturally extends the standard translation of modal logic into first-order formulas with one free variable, seen as the point of reference for the interpretation of the property expressed by the modal formula. Polyadic formulas get interpreted in tuples of states, hence they can be seen as special first-order formulas with several free variables. The arity of a polyadic formula is then the minimal number of free variables needed to express this property in FO+LFP or, equivalently, the length of the tuples used to interpret the formula.

The aim of this article is to show that the hierarchy formed by fragments of bounded arity, denoted $\mathcal{L}_\mu^1$, $\mathcal{L}_\mu^2$, ... is strict. This is not too surprising when taken literally: clearly, *any* satisfiable but non-valid formula in $\mathcal{L}_\mu^{k+1}$ is not equivalent to any formula in $\mathcal{L}_\mu^k$ since the former get interpreted in k+1-tuples and

*The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 105–116, doi:10.4204/EPTCS.191.10

the latter only in k-tuples. We therefore need to employ a convention that allows different fragments to be compared with respect to expressive power and still yields a meaningful hierarchy result. We consider formulas that are interpreted in a single state at the top-level, regardless of their arity. I.e. we show that for every $k \geq 1$ there is a $\mathcal{L}_\mu^{k+1}$-formula Φ_{k+1} such that there is no $\mathcal{L}_\mu^k$-formula ψ which yields

$$\mathcal{T}, (\underbrace{s, \ldots, s}_{k \text{ times}}) \models \psi \quad \text{iff} \quad \mathcal{T}, (\underbrace{s, \ldots, s}_{k+1 \text{ times}}) \models \Phi_{k+1}$$

for all labelled transition systems $\mathcal{T}$ and all their states s.

Arity hierarchies have been studied before, most notably by Grohe for fixpoint extensions of first-order logic including FO+LFP [10]. Even though each $\mathcal{L}_\mu^k$ can be embedded into FO+LFP, the arity hierarchy in $\mathcal{L}_\mu^\omega$ does not follow immediately from the one in FO+LFP. Grohe constructs formulas of arity $k+1$ in FO+LFP – they belong to the smaller FO+TC already – and shows that they are not equivalent to any formulas of arity k in FO+LFP – not even the much larger FO+sPFP. However, these witnessing formulas are not bisimulation-invariant since they express a relation formed by the transitive closure of a clique relation and being a clique is clearly not bisimulation-invariant. Hence, these witnessing formulas are not expressible in $\mathcal{L}_\mu^\omega$ and therefore the arity hierarchy is not transferred immediately.

It could of course be checked whether the proof used to show the arity hierarchy in FO+LFP could be adapted to work for $\mathcal{L}_\mu^\omega$ as well. It would require the search for a similar witnessing property and the adaption of the Ehrenfeucht-Fraïssé argument to the polyadic μ-calculus. Such model comparison games exist for the modal μ-calculus [21] but using them to obtain inexpressibility results has proved to be quite difficult.

Instead we use a simple diagonalisation argument in order to obtain a strictness result regarding arity hierarchies. A k-ary formula φ can be seen syntactically as a labelled transition system $\mathcal{T}_\varphi$, roughly based on the syntax-tree representation. We can then define a k+1-ary formula that simulates the evaluation of φ on $\mathcal{T}_\varphi$ and accepts those $\mathcal{T}_\varphi$ which are not accepted by φ itself. Hence, we need to find a generic way of dualising the operators in φ. This is no particular problem, for instance, when one sees a disjunction then one needs to check *both* disjuncts, for a conjunction one only needs to check one of them. However, fixpoint formulas may hold or not because of infinite recursive unfoldings through fixpoint operators. This needs to be dualised as well, and the only way that we can see to do this is to equip the simulating formula with a fixpoint structure that is at least as rich as the one of the simulated formula. Consequently, we obtain an arity hierarchy relative to the alternation hierarchy. This does not happen for extensions of First-Order Logic since it is known that there is no alternation hierarchy: every FO+LFP formula can be expressed with a single least fixpoint operator only [12, 23]. The situation for modal logics is different: more fixpoint alternation generally gives higher expressive power, at least so in the modal μ-calculus [6], and presumably then so in $\mathcal{L}_\mu^\omega$ as well.

The rest of this paper is organised as follows. In Section 2 we recall the polyadic μ-calculus and necessary tools like fixpoint alternation and model checking games. In Section 3 we prove the hierarchy results, and in Section 4 we conclude with a discussion on further work.

2 The Polyadic μ-Calculus

Labelled Transition Systems. Let $\mathsf{Prop} = \{p, q, \ldots\}$ and $\mathsf{Act} = \{a, b, \ldots\}$ be two fixed, countably infinite sets of atomic propositions and action names. A labeled transition system (LTS) over Prop and Act is a tuple $\mathcal{T} = (S, \to, \lambda, s_I)$ where S is a set of states, $\to \subseteq S \times \mathsf{Act} \times S$ is the transition relation, $\lambda : S \to 2^{\mathsf{Prop}}$ labels the states with atomic propositions, and s_I is some designated starting state. We will write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \to$.

The Syntax of $\mathcal{L}_\mu^\omega$. Let $\mathsf{Var} = \{X, Y, \ldots\}$ be an infinite set of second-order variables. The syntax of the polyadic modal μ-calculus $\mathcal{L}_\mu^\omega$ is similar to that of the ordinary modal μ-calculus. However, modalities and propositions are relativised to a natural number pointing at a position in a tuple of states used to interpret the formula.

A *replacement* is a $\kappa : \mathbb{N} \to \mathbb{N}$ which acts like the identity function on almost all arguments. We write $\mathbb{N} \dashrightarrow \mathbb{N}$ to denote the space of all replacements. Such a replacement is then written as $\{\kappa(i_1) \leftarrow i_1, \ldots, \kappa(i_m) \leftarrow i_m\}$ when $i_1 < \ldots < i_m$ are all those indices for which we have $\kappa(i_j) \neq i_j$. We will sometimes allow ourselves to deviate from this and to use some shorter but equally intuitive notation for such functions. For instance $\{1 \leftrightarrow 2\}$ should denote the swap between 1 and 2, i.e. it abbreviates $\{2 \leftarrow 1, 1 \leftarrow 2\}$.

For technical convenience, we define the logic directly in positive normal form. Formulas are then given by the grammar

$$\varphi \quad ::= \quad p(i) \mid \neg p(i) \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle_i \varphi \mid [a]_i \varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid \kappa\varphi$$

where $p \in \mathsf{Prop}$, $a \in \mathsf{Act}$, $1 \geq i \in \mathbb{N}$ and κ is a replacement. We require that every second-order variable gets bound by a unique fixpoint quantifier μ or ν. Then for every formula φ there is a function fp_φ which maps each second-order variable X occurring in φ to its unique binding formula $fp_\varphi(X) = \eta X.\psi$.

The set $Sub(\varphi)$ of subformulas of φ is defined as usual, with $Sub(\mu X.\varphi) = \{\mu X.\varphi\} \cup Sub(\varphi)$ for instance.

Later we will use the abbreviation $\ell \to \varphi$ when ℓ is a literal $q(i)$ or $\neg q(i)$. This behaves like ordinary implication – note that we have defined the logic in positive normal form and can therefore not simply introduce implication via negation – for such formulas when seen as $\bar{\ell} \vee \varphi$ where $e\bar{l}l$ is the usual complementary literal to ℓ.

The *arity* of a formula φ, denoted $ar(\varphi)$ is the largest index i occurring in the operators $p(i)$, $\langle a \rangle_i$, $[a]_i$ and $\{\kappa\}$ in any of its subformulas. The fragment of arity k is $\mathcal{L}_\mu^k := \{\varphi \mid ar(\varphi) \leq k\}$. Hence, $\varphi := \nu X.\langle a \rangle_1 \{2 \leftrightarrow 1\}X$ has arity 2 and it therefore belongs to all fragments $\mathcal{L}_\mu^2$, $\mathcal{L}_\mu^3$, etc., because it defines a relation of arity 2 which can also be seen as a relation of higher arity in which the 3rd, 4th, etc. components of its tuples are simply unrestrained.

The Semantics of $\mathcal{L}_\mu^\omega$. Formulas of $\mathcal{L}_\mu^k$ are interpreted in k-tuples of states of a transition system $\mathcal{T} = (S, \to, \lambda, s_I)$. An interpretation $\rho : \mathsf{Var} \to 2^{S^k}$ is neede in order to define this inductively and give a meaning to formulas with free variables. For each $\mathcal{L}_\mu^k$-formula φ, $[\![\varphi]\!]_\rho^\mathcal{T}$ is a k-ary relation of states in $\mathcal{T}$, namely the relation defined by φ under the assumption that its free variables are interpreted by ρ.

$$\begin{aligned}
[\![p(i)]\!]_\rho^\mathcal{T} &:= \{(s_1, \ldots, s_k) \mid p \in \lambda(s_i)\} \\
[\![\neg p(i)]\!]_\rho^\mathcal{T} &:= \{(s_1, \ldots, s_k) \mid p \notin \lambda(s_i)\} \\
[\![X]\!]_\rho^\mathcal{T} &:= \rho(X) \\
[\![\varphi \vee \psi]\!]_\rho^\mathcal{T} &:= [\![\varphi]\!]_\rho^\mathcal{T} \cup [\![\psi]\!]_\rho^\mathcal{T} \\
[\![\varphi \wedge \psi]\!]_\rho^\mathcal{T} &:= [\![\varphi]\!]_\rho^\mathcal{T} \cap [\![\psi]\!]_\rho^\mathcal{T} \\
[\![\langle a \rangle_i \varphi]\!]_\rho^\mathcal{T} &:= \{(s_1, \ldots, s_k) \mid \exists t \text{ s.t. } s_i \xrightarrow{a} t \text{ and } (s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k) \in [\![\varphi]\!]_\rho^\mathcal{T}\} \\
[\![[a]_i \varphi]\!]_\rho^\mathcal{T} &:= \{(s_1, \ldots, s_k) \mid \forall t : \text{ if } s_i \xrightarrow{a} t \text{ then } (s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k) \in [\![\varphi]\!]_\rho^\mathcal{T}\} \\
[\![\mu X.\varphi]\!]_\rho^\mathcal{T} &:= \bigcap \{R \subseteq S^k \mid [\![\varphi]\!]_{\rho[X \mapsto R]}^\mathcal{T} \subseteq R\} \\
[\![\nu X.\varphi]\!]_\rho^\mathcal{T} &:= \bigcup \{R \subseteq S^k \mid [\![\varphi]\!]_{\rho[X \mapsto R]}^\mathcal{T} \supseteq R\} \\
[\![\kappa\varphi]\!]_\rho^\mathcal{T} &:= \{(s_{\kappa(1)}, \ldots, s_{\kappa(k)}) \mid (s_1, \ldots, s_k) \in [\![\varphi]\!]_\rho^\mathcal{T}\}
\end{aligned}$$

Note that the partial order $\subseteq$ makes S^k a complete lattice with meets and joins given by $\bigcap$ and $\bigcup$, and the semantics of fixpoint formulas is then well-defined according to the Knaster-Tarski Theorem [13, 22].

We also write $\mathcal{T}, s_1, \ldots, s_k \models_\rho \varphi$ instead of $(s_1, \ldots, s_k) \in [\![\varphi]\!]^{\mathcal{T}}_\rho$. If φ has no free second-order variables then we also drop ρ. In Section 3 we will often consider situations with tuples of the form $(s, \ldots, s)$ of some length k derivable from the context. We will then simply write $\mathcal{T}, s \models \varphi$ as a short form for $\mathcal{T}, s, \ldots, s \models \varphi$.

Two formulas $\varphi, \psi \in \mathcal{L}^k_\mu$ are *equivalent*, written $\varphi \equiv \psi$, if $[\![\varphi]\!]^{\mathcal{T}}_\rho = [\![\psi]\!]^{\mathcal{T}}_\rho$ for any $\mathcal{T}$ and corresponding variable interpretation ρ. Note that two formulas can be equivalent even if they do not belong to the same arity fragment: if $\varphi \in \mathcal{L}^k_\mu$ and $\psi \in \mathcal{L}^{k'}_\mu$ and $k \neq k'$ then $\varphi, \psi \in \mathcal{L}^{\max\{k,k'\}}_\mu$, i.e. we can interpret the one of smaller arity as a formula of larger arity that simply does not constrain the additional elements in the tuples of the relation it defines.

Examples. The standard example of a $\mathcal{L}^\omega_\mu$-formula, indeed a $\mathcal{L}^2_\mu$-formula, is the one defining *bisimilarity*.

$$\varphi_\sim \;:=\; \nu X.\Big(\bigwedge_{p \in \mathsf{Prop}} p(1) \to p(2)\Big) \wedge \Big(\bigwedge_{a \in \mathsf{Act}} [a]_1 \langle a \rangle_2 X\Big) \wedge \{1 \leftrightarrow 2\}X$$

It is indeed the case that $\mathcal{T}, s, t \models \varphi_\sim$ iff $s \sim t$, i.e. s and t are bisimilar in $\mathcal{T}$.

As a second example consider an $\mathcal{T}$ with an edge relation flight and two atomic propositions warm and safe. When seeing the nodes of the LTS as cities (which can or cannot be warm and/or safe and are potentially linked by direct flight connections), then

$$\{3 \leftarrow 1\}\varphi_\sim \wedge \langle \mathsf{flight} \rangle_2 \mu X.\mathsf{warm}(2) \wedge \mathsf{safe}(2) \wedge \langle \mathsf{flight} \rangle_1 \varphi_\sim \wedge (\{3 \leftarrow 1\}\varphi_\sim \vee [\mathsf{flight}]_2 X) \wedge \{2 \leftarrow 3\}X$$

yields all triples (s, t, u) of cities such that there is a roundtrip from t which only traverses through warm and safe cities that can be reached from city s in one step – in case someone in s wants to come and visit – such that the trip can be traversed in both directions. This description of course uses equality ("roundtrip") on cities which is not available in the logic. Instead we use bisimilarity in the formula, so for instance "roundtrip from t" is to be understood as a trip starting in t and ending in a city that cannot be distinguished from t with the means of bisimilarity.

Fixpoint Alternation. The proof of the arity hierarchy carried out in Section 3 needs a closer look at the dependencies of fixpoints inside a formula. This phenomenon is well-understood leading to the notion of alternation hierarchy [9, 18]. We give a brief intoduction to fixpoint alternation that is sufficient for the purposes of the next section.

Let $k \geq 1$ and $\varphi \in \mathcal{L}^k_\mu$ be fixed. For two variables $X, Y \in Sub(\varphi)$ we write $X \geq_\varphi Y$ if X has a free occurrence in $fp_\varphi(Y)$. We use $>_\varphi$ to denote the strict part of its transitive closure. E.g. in

$$\varphi := \mu X.p(2) \vee \langle b \rangle_1 (\nu Y.q(1) \wedge \nu Y'.(\mu Z.Y' \vee \langle a \rangle_1 Z) \wedge [b]_2 Y)$$

we have $X >_\varphi Y >_\varphi Y' >_\varphi Z$ even though there is no free occurrence of X in the fixpoint formula for Z.

Names of variables do not matter but their fixpoint types do. So we abstract this chain of fixpoint dependencies into a chain $\mu >_\varphi \nu >_\varphi \nu >_\varphi \mu$. The *alternation type* of a formula is a maximal descending chain of variables (represented by their fixpoint types) such that adjacent types in this chain are different. The alternation type of φ above is therefore just (μ, ν, μ). We then define the *alternation hierarchy* as follows: Σ^k_m, respectively Π^k_m consists of all formulas of arity k and alternation type of length at most m such that the m-th last in this chain is μ, respectively ν, if it exists. For instance, the formula φ above belongs to Σ^2_3 and thefore also to Σ^2_m and Π^2_m for all $m > 3$. It does not belong to Π^2_2.

Each variable X occuring in φ is also given an *alternation depth* $ad_\varphi(X)$. It is the index in a maximal chain of dependencies $X_m >_\varphi \ldots >_\varphi X_1$ such that adjacent variables have different fixpoint types. E.g. in the example above we have $ad_\varphi(X) = 3$, $ad_\varphi(Y) = ad_\varphi(Y') = 2$ and $ad_\varphi(Z) = 1$.

The next observation is easy to see.

Lemma 1. *Let $\varphi \in \Sigma^k_m$ and $X \in Sub(\varphi)$ be one of its fixpoint variables. Then the fixpoint type of X is uniquely determined by $ad_\varphi(X)$, namely it is μ if m and i are both odd or both even, otherwise it is ν.*

Model Checking Games. We briefly recall model checking games for the polyadic μ-calculus [15]. They are defined in the same style as the model checking games for the modal μ-calculus [20] as a game played between players Verifier and Refuter on the product space of an LTS and a formula. Such games can be used to reason about the satisfaction of a formula in a structure since both satisfaction and non-satisfaction are reduced to the existence of winning strategies for one of the players in these model checking games.

As with the modal μ-calculus games, the model checking games for the polyadic μ-calculus are nothing more than parity games. However, they are played using k pebbles in the LTS and one pebble on the set of subformulas of the input formula. Hence, a configuration is a $k+1$-tuple written $s_1, \ldots, s_k \vdash \psi$ where the s_i are states of the underlying LTS $\mathcal{T} = (S, \to, \lambda, s_I)$ and ψ is a subformula of the underlying formula φ.

The rules are as follows.

- In a configuration of the form $s_1, \ldots, s_k \vdash \psi_1 \vee \psi_2$, player Verifier chooses an $i \in \{1, 2\}$ and the play continues with $s_1, \ldots, s_k \vdash \psi_i$. Intuitively, Verifier moves the formula pebble to a disjunct from the current disjunction.
- Likewise, in a configuration of the form $s_1, \ldots, s_k \vdash \psi_1 \wedge \psi_2$, player Refuter chooses such an i. Here, this can be seen as refuter moving the formula pebble.
- In a configuration of the form $s_1, \ldots, s_k \vdash \langle a \rangle_i \psi$, player Verifier chooses a t such that $s_i \xrightarrow{a} t$ and the play continues with $s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k \vdash \psi$. Intuitively, Verifier moves the i-the state pebble along an outgoing a-transition. The other $k-1$ pebbles that are on states remain where they are. The formula pebble is also moved to the next subformula.
- Likewise, in a configuration of the form $s_1, \ldots, s_k \vdash [a]_i \psi$, player Refuter chooses such a t.
- In a configuration of the form $s_1, \ldots, s_k \vdash \eta X.\psi$ or $s_1, \ldots, s_k \vdash X$ such that $fp_\varphi(X) = \eta X.\psi$, the formula pebble is simply moved to ψ, i.e. the play continues with $s_1, \ldots, s_k \vdash \psi$.

A player wins a play if the opponent cannot carry out a move anymore. Moreover, Verifier wins a play that reaches a configuration of the form $s_1, \ldots, s_k \vdash q(i)$ if $q \in \lambda(s_i)$. If, on the other hand, $q \notin \lambda(s_i)$ then player Refuter wins this play. Finally, there are infinite plays, and the winner is determines by the necessarily unique outermost fixpoint variable (i.e. the largest with respect to $>_\varphi$) that occurs infinitely often in this play. If its fixpoint type is ν, then Verifier wins, otherwise it is μ and Refuter wins.

The main advantage of these model checking games is the characterisation of the satisfaction relation via winning strategies in parity games (which they essentially are).

Proposition 2 ([15]). *Player* Verifier *has a winning strategy in the game in $\mathcal{T}$ and a closed φ, starting in the configuration $s_1, \ldots, s_k \vdash \varphi$ iff $\mathcal{T}, s_1, \ldots, s_k \models \varphi$.*

3 The Arity Hierarchy

3.1 The Principle Construction

The aim of this section is to show that $\mathcal{L}^1_\mu, \mathcal{L}^2_\mu, \ldots$ forms a strict hierarchy with respect to expressive power. The principles underlying the proof are easily explained: first we associate with each $\mathcal{L}^k_\mu$-formula φ an LTS $\mathcal{T}_\varphi$ with a designated starting state which we also call φ. Then we construct a closed $\mathcal{L}^{k+1}_\mu$-formula that, when given a $\mathcal{T}_\varphi$, reads off what φ is from $\mathcal{T}_\varphi$ and simulates its evaluation on it, checking that it does *not* hold on itself.

We first present the constructions principally, then discuss what results are achieved with the details of these constructions, and finally optimise the constructions such that the desired hierarchy result is achieved. We use a singleton Act wich means that we simply write $s \to t$ instead of $s \xrightarrow{a} t$ for the single action name 'a'. Likewise, we write $\Diamond_i$ and $\Box_i$ instead of $\langle a \rangle_i$ and $[a]_i$.

Construction of $\mathcal{T}_\varphi$. Let $k \geq 1$ be fixed and take an arbitrary closed $\varphi \in \mathcal{L}^k_\mu$. We assume that the set of propositions underlying φ is $\mathsf{Prop} = \{q_0, q_1, q_2, \ldots\}$. The construction of $\mathcal{T}_\varphi$ is largely based on the syntax-tree, respectively syntax-DAG of φ. We have $\mathcal{T}_\varphi = (Sub(\varphi), \to, \lambda, \varphi)$ with transitions given as follows.

$\psi_1 \odot \psi_2 \to \psi_i$	for every $\psi_1 \odot \psi_2 \in Sub(\varphi), \odot \in \{\wedge, \vee\}$ and every $i \in \{1,2\}$
$\odot\psi \to \psi$	for every $\odot\psi \in Sub(\varphi), \odot \in \{\Diamond_i, \Box_i, \kappa\}$ and every $i \in \{1, \ldots, k\}$
$\eta X.\psi \to \psi$	for every $\eta X.\psi \in Sub(\varphi)$ and $\eta \in \{\mu, \nu\}$
$X \to \psi$	for every $X \in Sub(\varphi)$ such that $fp_\varphi(X) = \eta X.\psi$

Thus, the graph structure of $\mathcal{T}_\varphi$ is indeed almost the one of the syntax-DAG of φ except for additional edges from fixpoint variables to their defining fixpoint formula.

The labelling of the nodes in $\mathcal{T}_\varphi$ remains to be defined. Remember that the ultimate goal is to construct a formula Φ^{k+1} which simulates the evaluation of φ on $\mathcal{T}_\varphi$. We will use k pebbles in order to simulate the k pebbles used in φ, and one additional pebble in order to store the subformula that is currently in question. Note that the satisfaction of a (closed) formula on an LTS only depends on the satisfaction of its subformulas. The position of this additional pebble will determine which subformula is currently evaluated. We therefore need to make the kind of subformula at a node in $\mathcal{T}_\varphi$ visible to a formula that is interpreted over it. This is what the state labels will be used for. Let

$$\mathsf{Prop}_0 := \{p^+_{j,i}, p^-_{j,i} \mid 1 \leq i \leq k, j \in \mathbb{N}\} \cup \{p^\wedge, p^\vee\} \cup \{p^\Diamond_i, p^\Box_i \mid 1 \leq i \leq k\} \cup \{p^{\mathsf{FP}}_i \mid 0 \leq i \leq m\} \cup \{p^{\mathsf{rp}}_\kappa \mid \kappa \in \mathbb{N} \dashrightarrow \mathbb{N}\}\,.$$

The labelling in $\mathcal{T}_\varphi$ is given as follows. Note that Prop is countably infinite.

$p^+_{j,i} \in \lambda(q_j(i))$	for every positive literal $q_j(i) \in Sub(\varphi)$
$p^-_{j,i} \in \lambda(q_j(i))$	for every negative literal $\neg q_j(i) \in Sub(\varphi)$
$p^\wedge \in \lambda(\psi_1 \wedge \psi_2)$	for every $\psi_1 \wedge \psi_2 \in Sub(\varphi)$
$p^\vee \in \lambda(\psi_1 \vee \psi_2)$	for every $\psi_1 \vee \psi_2 \in Sub(\varphi)$
$p^\Diamond_i \in \lambda(\Diamond_i\psi)$	for every $\Diamond_i\psi \in Sub(\varphi), 1 \leq i \leq k$
$p^\Box_i \in \lambda(\Box_i\psi)$	for every $\Box_i\psi \in Sub(\varphi), 1 \leq i \leq k$
$p^{\mathsf{rp}}_\kappa \in \lambda(\kappa\psi)$	for every $\kappa\psi \in Sub(\varphi), \kappa : \mathbb{N} \dashrightarrow \mathbb{N}$

$$p_i^{\mathsf{FP}} \in \lambda(\eta X.\psi), \lambda(X) \qquad \text{for every } \eta X.\psi, X \in Sub(\varphi), \eta \in \{\mu,\nu\} \text{ with } ad_\varphi(X) = i$$

With those labels a formula can see what the subformula at a node is that it is interpreted over, for instance whether it is a formula with a replacement modality as the principle operator, etc.

The construction of the simulating formulas. Next we construct formulas that simulate a φ on its own LTs representation $\mathcal{T}_\varphi$ and check that they do not satisfy themselves. The trick is simple: if $\varphi \in \mathcal{L}_\mu^k$ then we use k pebbles to simulate what φ would do with its k pebbles, and one additional pebble to check wich subformula we are currently evaluating. We let this pebble move through the syntax-DAG in a form that is dual to the semantics of the actual operators in the underlying φ; for instance in a conjunction we look for one conjunct, in a disjunction we continue with both disjuncts. We will use several fixpoint variables to dualise the fixpoint condition similar to the way it is done in the Walukiewcz formulas that express the winning conditions in parity games [24].

Let $m \geq 0$ and $k \geq 1$ be fixed. We construct a formula $\Phi_m^{k+1} \in \mathcal{L}_\mu^{k+1}$ as follows.

$$\begin{aligned}
\Phi_m^{k+1} \;:=\; \nu X_m.\mu X_{m-1}\ldots\eta X_1.\Big(& \bigwedge_{i=1}^{k}\bigwedge_{j\in\mathbb{N}} p_{j,i}^{+}(k+1) \to \neg q_j(i) \\
\wedge\; & \bigwedge_{i=1}^{k}\bigwedge_{j\in\mathbb{N}} p_{j,i}^{-}(k+1) \to q_j(i) \\
\wedge\; & p^{\wedge}(k+1) \to \Diamond_{k+1} X_1 \\
\wedge\; & p^{\vee}(k+1) \to \Box_{k+1} X_1 \\
\wedge\; & \bigwedge_{i=1}^{k} p_i^{\Diamond}(k+1) \to \Box_i \Box_{k+1} X_1 \\
\wedge\; & \bigwedge_{i=1}^{k} p_i^{\Box}(k+1) \to \Diamond_i \Box_{k+1} X_1 \\
\wedge\; & \bigwedge_{\kappa\in\mathbb{N}\dashrightarrow\mathbb{N}} p_\kappa^{\mathsf{rp}}(k+1) \to \kappa \Box_{k+1} X_1 \\
\wedge\; & \bigwedge_{i=1}^{m} p_i^{\mathsf{FP}}(k+1) \to \Box_{k+1} X_i \Big)
\end{aligned}$$

where $\eta = \nu$ if m is odd and $\eta = \mu$ otherwise.

Remark 1. Of course, Φ_m^{k+1} is not a formula strictly speaking because of the potentially infinite conjunctions in the first two clauses. There is an easy way to fix this: we assume a finite set of atomic propositions $\{p, q, \ldots\}$. Then a finite conjunction obviously suffices and Φ_m^{k+1} is indeed a formula. However, we need to address the issue of choice of atomic propositions in Section 3.2 below anyway. So for the moment we simply accept the small flaw about infinite conjunctions as an intermediate step and as a means to separate the principles from the details in this construction.

Note that this problem does not arise in the clause with the p_κ^{rp} since k is fixed, and κ can at most change the first k pebbles. Hence, there are only finitely many such κ.

We need two observations about Φ_m^{k+1}. The first, a syntactic one, is easy to verify.

Lemma 3. *For every $m \geq 0$ and every $k \geq 1$ we have $\Phi_m^{k+1} \in \Pi_m^{k+1}$.*

The second one is of a semantic nature and states that Φ_m^{k+1} does what it is supposed to do.

Lemma 4. *Let $m \geq 0$, $k \geq 1$ and $\varphi \in \Sigma_m^k$. Then we have $\mathcal{T}_\varphi, \varphi \models \Phi_m^{k+1}$ iff $\mathcal{T}_\varphi, \varphi \not\models \varphi$.*

Proof. We argue using model checking games for $\mathcal{L}_\mu^\omega$.

"$\Leftarrow$" Suppose we have $\mathcal{T}_\varphi, \varphi \not\models \varphi$, i.e. Refuter has a winning strategy for the game $\mathcal{G}$ played on $\mathcal{T}_\varphi$, k pebbles initially placed on the node φ in it, and the $\mathcal{L}_\mu^k$-formula φ itself. This gives rise to a strategy for player Verifier in the game $\mathcal{G}'$ played on $\mathcal{T}_\varphi$, now $k+1$ pebbles placed on node φ initially, and the formula Φ_m^{k+1}. The fact that each node in $\mathcal{T}_\varphi$ satisfies exactly on atomic proposition of the kind p^* and at most one i or at most on κ means that any play which Refuter does not lose immediately selects a clause in Φ_m^{k+1}, carries out some operation on the pebbles and then loops through some fixpoint variable.

It is not hard to see that Verifier can use Refuter's strategy from $\mathcal{G}$ to follow the operations carried out on the pebbles prescribed by each clause without losing. For instance, if the third clause demands her to choose a successor for the $k+1$-st pebble then she takes the one that represents the conjunct that Refuter would chose in the same situation in $\mathcal{G}$. This way, every play in $\mathcal{G}'$ that conforms to her strategy has an underlying play in $\mathcal{G}$ that conforms to Refuter's strategy there. If that one is won by Refuter because Verifier got stuck at some point then this can only be because the play reached a position of the form $(s_1, \ldots, s_k) \vdash \Diamond_i \psi$ and s_i has no successor. In the corresponding play in $\mathcal{G}'$, pebble $k+1$ will be on a node with label $p_i^\Diamond$, and this requires Refuter to move the i-th pebble to a successor which equally he cannot. Notice that the clause with $p_i^\Diamond$ contains the operator $\Box$ and vice-versa. Thus, Verifier wins the corresponding play in $\mathcal{G}'$.

Suppose that the underlying play in $\mathcal{G}$ is won by Refuter because the largest fixpoint variable X that is seen infinitely often is of type μ. Then we must have $ad_\varphi(X) = i$ for some i, and then the play in $\mathcal{G}'$ will infinitely often go through positions that are labelled with p_i^{FP}, and it will eventually not go through positions that are labelled with $p_{i'}^{\mathsf{FP}}$ with $i' > i$ anymore. All that remains to be seen in this case is that the largest variable seen infinitely often in the play on Φ_m^{k+1} is of type ν. This is a direct consequence of Lemma 1. Hence, Verifier wins such plays, too, which shows that her strategy derived from Refuter's winning strategy in $\mathcal{G}$ is winning for her in $\mathcal{G}'$.

"$\Rightarrow$" This is shown by contraposition in the same way now assuming a winning strategy for Verifier in the game on $\mathcal{T}_\varphi$ and φ and turning it into a winning strategy for Refuter in the game on $\mathcal{T}_\varphi$ and Φ_m^{k+1}.

Lemma 5. *Let $m \geq 0$ and $k \geq 1$. There is no $\varphi \in \Sigma_m^k$ such that $\varphi \equiv \Phi_m^{k+1}$.*

Proof. Suppose there was such a φ. Then we would have

$$\mathcal{T}_\varphi, \varphi \models \varphi \quad \text{iff} \quad \mathcal{T}_\varphi, \varphi \models \Phi_m^{k+1} \quad \text{iff} \quad \mathcal{T}_\varphi, \varphi \not\models \varphi$$

first because of the assumed equivalence and second because of Lemma 4.

Thus, we could summarise the findings from these lemmas and also uses the observation that the entire construction is equally possible for formula in Π_m^k then yielding a $\Phi_m^{k+1} \in \Sigma_m^{k+1}$. Then we get that for all $m \geq 0$ and $k \geq 1$ we have $\Sigma_m^k \not\supseteq \Pi_m^{k+1}$ and $\Pi_m^k \not\supseteq \Sigma_m^{k+1}$. Consequently, we have $\Sigma_m^k \subsetneq \Sigma_{m+1}^{k+1}$ and $\Pi_m^k \subsetneq \Pi_{m+1}^{k+1}$.

The reason why we do not formally state this as a theorem (yet) is discussed next.

3.2 The Hierarchy over a Fixed Small Signature

Consider what is happing with the set of atomic propositions in the construction of the previous Section 3.1. We have already seen in Remark 1 that the construction does not work for an infinite set of

atomic propositions Prop. Even if this is finite, then the construction does work but it has the following effect: we simulate a formula with k pebbles over Prop by a formula with $k+1$ pebbles over $\mathsf{Prop} \cup \mathsf{Prop}_0$. It is not surprising that we obtain formulas over this extended signature which cannot be expressed over the smaller one. In order to argue that the hierarchy of inexpressibility as laid out in the previous section is truly meaningful we would need $\mathsf{Prop} = \mathsf{Prop} \cup \mathsf{Prop}_0$ or, at least, that the two sets have equal cardinality so that some bijection between them could be used as an encoding.

In the following we will show how the construction can be fixed such that it works over a fixed finite set

$$\mathsf{Prop}_1 := \{p^+, p^-, p^\wedge, p^\vee, p^\Diamond, p^\Box, p^{\mathsf{FP}}, p^{\mathsf{rp}}, p^{\mathsf{sw}}, p^\bullet\}$$

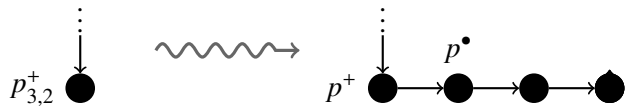
of atomic propositions. Thus, we do not encore the index of propositions, the level in the fixpoint hierarchy, and the kind of operation on pebbles in those propositions anymore. Instead we will encode this missing information in the graph structure of $\mathcal{T}_\varphi$ (rather than in its labels). For the replacement modalities κ we need a little preparation.

A replacement κ is called simple if it is of the form $\{i \leftarrow j\}$ or $\{i \leftrightarrow j\}$. A formula is called *normalised* if every replacement in it is simple. The following is a simple consequence of the fact that every function $\kappa : \mathbb{N} \dashrightarrow \mathbb{N}$ that leaves all numbers greater than k untouched, can be constructed by a sequence of swaps between $i, j \le k$, followed by some simple mappings from some i to a j.

Lemma 6. *Let $m \ge 0$, $k \ge 1$. Every $\varphi \in \Sigma^k_m$, respectively Π^k_m is equivalent to a normalised $\varphi' \in \Sigma^k_m$, respectively Π^k_m.*

We therefore assume that from now on, all formulas φ to be simulated are normalised. We change the construction of $\mathcal{T}_\varphi$ as follows.

1. Suppose there is a state s of the form $q_j(i)$ or $\neg q_j(i)$, necessarily labeled with $p^+_{j,i}$ or $p^-_{j,i}$. Replace the proposition by p^+, respectively p^-, and add a new finite path of length $i + j$ to this node such that the $(i-1)$-st new state has the label $p^\bullet$.



   Let $\kappa^{\leftarrow}_{\mathsf{cyc}} := \{k \leftarrow 1, 1 \leftarrow 2, \ldots, k-1 \leftarrow k\}$ and consider the formula

$$\mathit{searchPeb} := (p^\bullet \wedge q_j(1)) \vee \kappa^{\leftarrow}_{\mathsf{cyc}} \Diamond_{k+1}((p^\bullet \wedge q_j(1)) \vee \kappa^{\leftarrow}_{\mathsf{cyc}} \Diamond_{k+1}((p^\bullet \wedge q_j(1)) \vee \ldots \kappa^{\leftarrow}_{\mathsf{cyc}} \Diamond_{k+1}(p^\bullet \wedge q_j(1)) \ldots)) \tag{1}$$

   with $k-1$ occurrences of $\kappa^{\leftarrow}_{\mathsf{cyc}}$. It is true in s at pebble $k+1$ with the additional path iff $q_j(i)$ was true in s with the original construction. The first part of this new path is used to shift the pebbles until the i-th has become the first and then, instead of checking whether the i-th pebble is on a state satisfiying q_j, we can now check the first one instead. Note that this formulas moves pebble number $k+1$ along this new path but the other k pebbles remain where they are; apart from being cyclically changed around.

   We can of course equally construct such a formula that mimicks the checking of $\neg q_j(i)$.

   Finally, we also need to use the remaining path to read off the encoding of j. This can easily be done as follows.

$$\mathit{searchProp}_j := q_0(1) \vee \Diamond_{k+1}(q_1(1) \vee \Diamond_{k+1}(q_2(1) \vee \ldots \Diamond_{k+1}(q_{h-2}(1) \vee \Diamond_{k+1} q_{h-1}(1)) \ldots))$$

This formula is then used instead of $q_j(1)$ in (1), and the resulting formula is used instead of $q_j(i)$ in the clause for $p^-_{j,i}$ in Φ^{k+1}_m. Hence, this clase simply becomes

$$\ldots \wedge p^-_{j,i} \to searchPeb[searchProp/q_j(1)]$$

where $\psi[\chi/\chi']$ denotes the formula that results from ψ by replacing every subformula χ' with χ.

2. An edge of the form $\eta X.\psi \to \psi$ or $X \to \psi$ is replaced in similar style by a sequence of i edges, marking the last state after them with $p^\bullet$. Then we can replace the label p_i^{FP} with p^{FP} in the first state, and the corresponding clause in Φ^{k+1}_m with

$$\ldots \wedge p^{\mathsf{FP}} \to \Box_{k+1}((p^\bullet \wedge X_1) \vee \Box_{k+1}((p^\bullet \wedge X_2) \vee \Box_{k+1}(\ldots \vee \Box_{k+1}(p^\bullet \wedge X_m)\ldots)))$$

3. An edge of the form $\Diamond_i\psi \to \psi$ or $\Box_i\psi \to \psi$ is replaced by a sequence of $2i$ edges via new states, and $p^\bullet$ must hold after i and after $2i$ steps. The trick to use here is to cycle the first k pebbles until the i-th one becomes the first, then execute the corresponding action for the i-pebble on the first one instead, and then cycle them back again. Let $\kappa^{\leftarrow}_{\mathsf{cyc}}$ be as above and $\kappa^{\rightarrow}_{\mathsf{cyc}} := \{2 \leftarrow 1, 3 \leftarrow 2, \ldots, 1 \leftarrow k\}$. Then we can replace the clause for $p_i^\Diamond$ in Φ^{k+1}_m by

$$\ldots \wedge p^\Diamond \to (p^\bullet \wedge \Box_{k+1}(\Box_1 goBack \vee \kappa^{\leftarrow}_{\mathsf{cyc}}(p^\bullet \wedge \Box_{k+1}(\Box_1 goBack \vee \ldots (p^\bullet \wedge \Box_{k+1}\Box_1 goBack)\ldots)))$$

with exactly $k-1$ occurrences of $\kappa^{\leftarrow}_{\mathsf{cyc}}$ and

$$goBack := \Box_{k+1}((p^\bullet \wedge X_1) \vee \kappa^{\rightarrow}_{\mathsf{cyc}}\Box_{k+1}((p^\bullet \wedge X_1) \vee \ldots \Box_{k+1}(p^\bullet \wedge X_1)\ldots))$$

with exactly $k-1$ occurrences of $\kappa^{\rightarrow}_{\mathsf{cyc}}$.

Likewise, we can use the same trick to eliminate the dependence on i of the formula Φ^{k+1}_m in the clause for $p_i^\Box$ which is equally replaced by $p^\Box$, and those paths of length $2i$ can be used to decode the value i from the graph structure instead of reading it straight off the atomic proposition.

4. Finally, we can use the same trick in a slightly more elaborate fashion to handle replacement modailities of the form $\{i \leftarrow j\}$ and $\{i \leftrightarrow j\}$. We mark nodes in $\mathcal{T}_\varphi$ that correspond to the form by p^{rp} and those that correspond to the latter by p^{sw}. A swap of the form $\{i \leftrightarrow j\}$ can be handled as follows: assume $i < j$.
   (a) Cyclically shift the pebbles $1, \ldots, k$ for i positions to the left.
   (b) Cyclically shift the pebbles $2, \ldots, k$ for $j-i-1$ positions to the left.
   (c) Swap pebbles 1 and 2.
   (d) Cyclically shift the pebbles $2, \ldots, k$ for $j-i-1$ positions to the right.
   (e) Cyclically shift the pebbles $1, \ldots, k$ for i positions to the right.

   Hence, we replace a transition of the form $\{i \leftarrow j\}\psi \to \psi$ by a path of length $2j-2$ and mark the states at positions i, $j-1$, $2j-2$ and the last one with $p^\bullet$ so that we can, like above, construct a formula that mimicks the five steps above to carry out the swapping of pebbles i and j.

   The construction for replacements of the form $\{i \leftarrow j\}$ is similar. Again, the trick is to cycle i and j to positions 1 and 2, carry the replacement out on these fixed positions, and cycle the pebbles back again. These eliminates the dependence of Φ^{k+1}_m on propositions which carry such a value.

With this being done, Φ_m^{k+1} becomes a formula that is defined over a fixed set Prop_1 of atomic propositions of size 10, and we can use it to simulate formulas $\varphi \in \Sigma_m^k$ over the same Prop_1. Then the inexpressibility result of the previous section becomes meaningful. Using standard encoding techniques we can break the resul down to $\mathcal{L}_\mu^\omega$ over two atomic propositions only, using binary encoding, or a single one, using unary encoding. The atomic propositions can also be eliminated entirely by appending certain finite trees to the states in which they hold such that these trees are checkable using fixpoint-free formulas of $\mathcal{L}_\mu^\omega$. Hence, we get the following.

Theorem 7. *For all $m \geq 0$ and $k \geq 1$ we have $\Sigma_m^k \not\supseteq \Pi_m^{k+1}$ and $\Pi_m^k \not\supseteq \Sigma_m^{k+1}$. Consequently, we have $\Sigma_m^k \subsetneq \Sigma_{m+1}^{k+1}$ and $\Pi_m^k \subsetneq \Pi_{m+1}^{k+1}$. These results hold independently of the underlying signature* Prop *and* Act.

4 Conclusion and Further Work

We have shown that the arity hierarchy in the polyadic μ-calculus, a modal fixpoint logic for specifying bisimulation-invariant relational properties of states in transition systems, is strict in the sense that higher arity gives higher expressive power provided that one is allowed to use a little bit more fixpoint alternation ($\Sigma_m^k \subsetneq \Sigma_{m+1}^{k+1}$). If alternation must not increase then higher arity yields not necessarily more but different expressiveness ($\Sigma_m^k \not\supseteq \Pi_m^{k+1}$).

Obviously, the exact effects on expressive power that should be attributed to arity and to fixpoint alternation need to be separated. A first step would be to prove the strictness of the alternation hierarchy within each $\mathcal{L}_\mu^k$. For $k = 1$, i.e. the ordinary μ-calculus, this is known for arbitrary and in particular for finite transition systems [6, 17]. Subsequently, the result could be shown for several other classes of transition systems, for instance binary trees [3, 7], nested words [11] and graphs whose edge relation satisfies certain properties like being transitive for instance [8, 1].

We suspect that not only is the alternation hierarchy within each $\mathcal{L}_\mu^k$ also strict, but equally that Arnold's proof [3] using a similar diagonalisation argument for $\mathcal{L}_\mu$ can be extended. It relies on the interreducibility between model checking for $\mathcal{L}_\mu$ and parity games [20] and in particular the existence of the Walukiewicz formulas defining winning regions in parity games [24]. It is known [15] that the model checking problem for $\mathcal{L}_\mu^k$ and any $k \geq 1$ can equally be reduced to a parity game, and it seems feasible to extend the construction of the Walukiewicz formulas to higher arity. This would use similar principles as those underlying the construction of Φ_m^{k+1} in Section 3.

Model checking $\mathcal{L}_\mu^k$ can also be reduced to model checking $\mathcal{L}_\mu$ directly using k-products of transition systems, i.e. there is a translation of $\mathcal{L}_\mu^k$-formulas to $\mathcal{L}_\mu$-formulas that preserves truth under taking k-fold products of transition systems [19, 15]. Hence, the question of the strictness of the alternation hierarchy in $\mathcal{L}_\mu^k$ is equivalent to the question after the strictness of the $\mathcal{L}_\mu$ alternation hierarchy over the class of all k-fold products of transition systems.

References

[1] L. Alberucci & A. Facchini (2009): *The modal μ-calculus over restricted classes of transition systems*. Journal of Symbolic Logic 74(4), pp. 1367–1400, doi:10.2178/jsl/1254748696.

[2] H. R. Andersen (1994): *A Polyadic Modal μ-Calculus*. Technical Report ID-TR: 1994-195, Dept. of Computer Science, Technical University of Denmark, Copenhagen, doi:10.1.1.42.1859.

[3] A. Arnold (1999): *The modal μ-calculus alternation hierarchy is strict on binary trees*. RAIRO - Theoretical Informatics and Applications 33, pp. 329–339, doi:10.1051/ita:1999121.

[4] J. Bradfield & C. Stirling (2001): *Modal logics and μ-calculi: an introduction*. In J. Bergstra, A. Ponse & S. Smolka, editors: *Handbook of Process Algebra*, Elsevier, pp. 293–330, doi:10.1016/B978-044482830-9/50022-9.

[5] J. Bradfield & C. Stirling (2007): *Modal mu-calculi*. In P. Blackburn, J. van Benthem & F. Wolter, editors: *Handbook of Modal Logic: Studies in Logic and Practical Reasoning Volume 3*, Elsevier, pp. 721–756, doi:10.1016/S1570-2464(07)80015-2.

[6] J. C. Bradfield (1996): *The Modal μ-calculus Alternation Hierarchy Is Strict*. In: *Proc. 7th Conf. on Concurrency Theory, CONCUR'96*, *LNCS* 1119, Springer, pp. 233–246, doi:10.1007/3-540-61604-7_58.

[7] J. C. Bradfield (1999): *Fixpoint Alternation: Arithmetic, Transition Systems, and the Binary Tree*. *RAIRO - Theoretical Informatics and Applications* 33(4/5), pp. 341–356, doi:10.1051/ita:1999122.

[8] G. D'Agostino & Giacomo Lenzi (2010): *On the μ-calculus over transitive and finite transitive frames*. *Theoretical Computer Science* 411(50), pp. 4273–4290, doi:10.1016/j.tcs.2010.09.002.

[9] E. A. Emerson & C. L. Lei (1986): *Efficient Model Checking in Fragments of the Propositional μ–Calculus*. In: *Symposion on Logic in Computer Science*, IEEE, Washington, D.C., USA, pp. 267–278.

[10] M. Grohe (1996): *Arity hierarchies*. *Annals of Pure and Applied Logic* 82(2), pp. 103–163, doi:10.1016/0168-0072(95)00072-0.

[11] J. Gutierrez, F. Klaedtke & M. Lange (2014): *The μ-Calculus Alternation Hierarchy Collapses over Structures with Restricted Connectivity*. *Theoretical Computer Science* 560(3), pp. 292–306, doi:10.1016/j.tcs.2014.03.027.

[12] N. Immerman (1986): *Relational Queries Computable in Polynomial Time*. *Information and Control* 68(1–3), pp. 86–104, doi:10.1016/S0019-9958(86)80029-8.

[13] B. Knaster (1928): *Un théorèm sur les fonctions d'ensembles*. *Annals Soc. Pol. Math* 6, pp. 133–134.

[14] D. Kozen (1982): *Results on the Propositional μ-Calculus*. In: *Proc. 9th Int. Coll. on Automata, Languages and Programming, ICALP'82*, *LNCS* 140, Springer, pp. 348–359, doi:10.1007/BFb0012782.

[15] M. Lange & E. Lozes (2012): *Model Checking the Higher-Dimensional Modal μ-Calculus*. In: *Proc. 8th Workshop on Fixpoints in Computer Science, FICS'12*, *Electr. Proc. in Theor. Comp. Sc.* 77, pp. 39–46, doi:10.4204/EPTCS.77.

[16] M. Lange, E. Lozes & M. Vargas Guzmán (2014): *Model-Checking Process Equivalences*. *Theoretical Computer Science* 560, pp. 326–347, doi:10.1016/j.tcs.2014.08.020.

[17] G. Lenzi (1996): *A Hierarchy Theorem for the μ-Calculus*. In: *Proc. 23rd Int. Coll. on Automata, Languages and Programming, ICALP'96*, *LNCS* 1099, Springer, pp. 87–97, doi:10.1007/3-540-61440-0_119.

[18] D. Niwiński (1988): *Fixed Points vs. Infinite Generation*. In: *Proc. 3rd Ann. Symp. on Logic in Computer Science, LICS'88*, IEEE Computer Society, pp. 402–409.

[19] M. Otto (1999): *Bisimulation-invariant PTIME and higher-dimensional μ-calculus*. *Theor. Comput. Sci.* 224(1–2), pp. 237–265, doi:10.1016/S0304-3975(98)00314-4.

[20] C. Stirling (1995): *Local Model Checking Games*. In: *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, *LNCS* 962, Springer, pp. 1–11, doi:10.1007/3-540-60218-6_1.

[21] C. Stirling (1996): *Games and Modal μ-Calculus*. In T. Margaria & B. Steffen, editors: *Proc. 2nd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'96*, *LNCS* 1055, Springer, pp. 298–312, doi:10.1007/3-540-61042-1_51.

[22] A. Tarski (1955): *A Lattice-theoretical Fixpoint Theorem and its Application*. *Pacific Journal of Mathematics* 5, pp. 285–309, doi:10.2140/pjm.1955.5.285.

[23] M. Y. Vardi (1982): *The Complexity of Relational Query Languages (Extended Abstract)*. In: *Proc. 14th Symp. on Theory of Computing, STOC'82*, ACM, San Francisco, CA, USA, pp. 137–146, doi:10.1145/800070.

[24] I. Walukiewicz (2002): *Monadic second-order logic on tree-like structures*. *Theor. Comput. Sci* 275(1-2), pp. 311–346, doi:10.1016/S0304-3975(01)00185-2.

Disjunctive form and the modal μ alternation hierarchy

Karoliina Lehtinen
Laboratory for Foundations of Computer Science
University of Edinburgh
M.K.Lehtinen@sms.ed.ac.uk

This paper studies the relationship between disjunctive form, a syntactic normal form for the modal μ calculus, and the alternation hierarchy. First it shows that all disjunctive formulas which have equivalent tableau have the same syntactic alternation depth. However, tableau equivalence only preserves alternation depth for the disjunctive fragment: there are disjunctive formulas with arbitrarily high alternation depth that are tableau equivalent to alternation-free non-disjunctive formulas. Conversely, there are non-disjunctive formulas of arbitrarily high alternation depth that are tableau equivalent to disjunctive formulas without alternations. This answers negatively the so far open question of whether disjunctive form preserves alternation depth. The classes of formulas studied here illustrate a previously undocumented type of avoidable syntactic complexity which may contribute to our understanding of why deciding the alternation hierarchy is still an open problem.

1 Introduction

The modal μ calculus [2], L_μ, is a modal logic augmented with its namesake least fixpoint operator μ and the dual greatest fixpoint operator, ν. Alternating between these two operators gives the logic its great expressivity [1] while both model checking and satisfiability remain pleasingly decidable. The complexity of model checking is, at least currently, tied to the number of such alternations, called the alternation depth of the formula being checked [10]. The problem of deciding the least number of alternations required to express a property, also known as the Rabin-Mostowski index problem, is a long standing open problem.

Disjunctive normal form is a syntactic restriction on L_μ formulas which first appeared in [9] and was then used as a tool for proving completeness of Kozen's axiomatization [14]. It is based on the tableau decomposition of a formula which forces it to be in many ways well-behaved, making it a useful tool for various manipulations. For instance, satisfiability and synthesis are straight-forward for disjunctive formulas. In [5] it is used to analyse modal L_μ from a logician's perspective. More recently, disjunctive form was found to allow for simple formula optimisation: if a formula is equivalent to a formula without greatest fixpoints, then such a formula is easily produced by simple syntactic manipulation on the disjunctive form of the formula [11].

Each of these results uses the fact that any formula can be effectively transformed into an equivalent disjunctive formula with the same tableau – indeed, disjunctive form is perhaps the closest one gets to a canonical normal form for L_μ. The transformation itself, described in [9], is involved and it has so far been an open question whether it preserves the alternation depth of formulas. If this was the case, it would be sufficient to study the long-standing open problem of the decidability of the alternation hierarchy on this well-behaved fragment.

In this paper, we show that although the disjunctive fragment of L_μ is itself well-behaved with respect to the alternation hierarchy, the transformation into it does not preserve alternation depth.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 117–131, doi:10.4204/EPTCS.191.11

The transformation into disjunctive form takes the tableau decomposition of a formula, and produces a disjunctive formula that generates the same tableau. The first contribution of this paper is to show that all disjunctive formulas generating the same tableau have the same alternation depth. This result brings some clarity to the transformation into disjunctive form since one of the more difficult steps of the construction is representing the parity of infinite paths of the tableau with a finite priority assignment. The result presented here means that all valid choices are just as good, as all yield a disjunctive formula of the same alternation depth. As a result, the alternation hierarchy is decidable for the disjunctive fragment of L_μ with respect to tableau equivalence, a stricter notion of equivalence than semantic equivalence, as defined in [14].

The second contribution of this paper is to show that this does not extend to non-disjunctive formulas. Not only does tableau equivalence not preserve alternation depth in general, but the alternation depth of a formula does not guarantee *any* upper bound on the alternation depth of equivalent disjunctive formulas. Indeed, for arbitrarily large n, there are formulas with a single alternation which are tableau equivalent only to disjunctive formulas with at least n alternations.

Conversely, there are formulas of L_μ with arbitrarily large alternation depth which are tableau equivalent to a disjunctive formula without alternations. This shows that the alternation depths of tableau equivalent formulas are only directly related within the disjunctive fragment.

The signficance of these results in twofold. First, they outline the limits of what can be achieved using disjunctive form: disjunctive form does not preserve alternation depth so despite being a useful tool for satisfiability-related problems, it is unlikely to be of much help in contexts where the alternation depth of a formula matters, such as model-checking or formula optimisation beyond the first levels of the alternation hierarchy.

Secondly, and perhaps most significantly, these results impact our understanding of the alternation hierarchy. This paper's results imply that deciding the alternation hierarchy for the disjunctive fragment of L_μ, an open but easier problem, is not sufficient for deciding the alternation hierarchy in the general case. The counterexamples used to show this illustrate a previously undocumented type of accidental complexity which appears to be difficult to identify. These may shed light on why deciding the alternation hierarchy is still an open problem and examplify a category of formulas with unnecessary alternations which need to be tackled with novel methods.

Related work Deciding the modal μ alternation hierarchy is exactly equivalent to deciding the Rabin-Mostowski index of alternating parity automata. The corresponding problem has also been studied for automata operating on words [3] and automata which are deterministic [13, 12], or non-deterministic [4, 7] rather than alternating. As will be highlighted throughout this paper, many of the methods used here are similar to methods applied to different types of automata.

2 Preliminaries

2.1 The modal μ calculus

For clarity and conciseness, the semantics of L_μ are given directly in terms of parity games. As is well documented in the literature, this approach is equivalent to the standard semantics [2]. The following definitions are fairly standard, although we draw the reader's attention to the use of the less typical modality $\rightarrow\mathscr{B}$ in the syntax of L_μ and the unusual but equivalent definition of alternation depth.

Definition 1. (L_μ) Given a set of atomic propositions $Prop = \{P, Q, ...\}$ and a set of fixpoint variables $Var = \{X, Y, ...\}$, the syntax of L_μ is given by:

$$\phi := \top \mid \bot \mid P \mid \neg P \mid X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \rightarrow\mathscr{B} \text{ where } \mathscr{B} \text{ is a set of formulas } \mid \mu X.\phi \mid \nu X.\phi$$

The modality $\rightarrow\mathscr{B}$ replaces the more usual modalities $\Diamond\phi$ and $\Box\phi$. If $\mathscr{B}$ is a set of formulas, $\rightarrow\mathscr{B}$ stands for $(\bigwedge_{\phi\in\mathscr{B}} \Diamond\phi) \wedge \Box \bigvee_{\phi\in\mathscr{B}} \phi$: every formula in $\mathscr{B}$ must be realised in some successor state and each successor state must realise at least one of the formulas in $\mathscr{B}$. The modalities $\Diamond\phi$ and $\Box\phi$ are expressed in this syntax by $\rightarrow\{\phi, \top\}$ and $\rightarrow\{\phi\} \vee \rightarrow\bot$ respectively, where $\bot$ denotes the empty set.

Without loss of expressivity, this syntax only allows for formulas in positive form: negation is only applied to propositions. Furthermore, without loss of expressivity, but perhaps conciseness, we require all formulas to be guarded: all fixpoint variables are within the scope of a modality within their binding formula. For the sake of clarity, we restrict our study to the uni-modal case but expect the multi-modal case to behave broadly speaking similarly. To minimise the use of brackets, the scope of fixpoint bindings should be understood to extend as far as possible.

Definition 2. *(Structures)* A structure $\mathscr{M} = (S, s_0, R, P)$ consists of a set of states S, rooted at some initial state $s_0 \in S$, and a successor relation $R \subseteq S \times S$ between the states. Every state s is associated with a set of propositions $P(s) \subseteq Prop$ which it is said to satisfy.

Definition 3. *(Parity games)* A parity game is a potentially infinite two-player game on a finite graph $\mathscr{G} = (V_0, V_1, E, v_I, \Omega)$ of which the vertices $V_0 \cup V_1$ are partitioned between the two players Even and Odd and annotated with positive integer priorities via $\Omega : V_0 \cup V_1 \rightarrow \mathbb{N}$. The even player and her opponent, the odd player, move a token along the edges $E \subseteq V_0 \cup V_1 \times V_0 \cup V_1$ of the graph starting from an initial position $v_I \in V_0 \cup V_1$, each choosing the next position when the token is on a vertex in their partition. Some positions p might have no successors in which case they are winning for the player of the parity of $\Omega(p)$. A play consists of the potentially infinite sequence of vertices visited by the token. For finite plays, the last visited parity decides the winner of the play. For infinite play, the parity of the highest priority visited infinitely often decides the winner of the game: Even wins if the highest priority visited infinitely often is even; otherwise Odd wins. Note that since some readers may be used to an equivalent definition using the lowest priority to define the winner, whenever possible, "most significant" will be used to indicate the highest priority.

Definition 4. *(Strategies)* A positional strategy σ for one of the players in $\mathscr{G} = (V_0, V_1, E, v_I, \Omega)$ is a mapping from the player's positions s, in V_0 for Even and in V_1 for Odd, in the game to a successor position s' such that $(s, s') \in E$. A play respects a player's strategy σ if the successor of any position in the play belonging to the player is the one dictated by σ. If σ is Even's strategy and τ is Odd's strategy then there is a unique play $\sigma \times \tau$ respecting both strategies from every position. The winner of the parity game at a position is the player who has a strategy σ , said to be a winning strategy, such that they win $\sigma \times \tau$ from that position for any counter-strategy τ. The following states that such strategies are sufficient: players do not need to take into account the history of a play to play optimally.

Fact 5. *Parity games are positionally determined: for every position either Even or Odd has a winning strategy [6].*

This means that strategies gain nothing from looking at the whole play rather than just the current position. As a consequence, we may take a strategy to be memoryless: it is a mapping from a player's positions to a successor.

For any L_μ formula ϕ and a structure $\mathscr{M}$ we define a parity game $\mathscr{M} \times \phi$, constructed in polynomial time, and say that $\mathscr{M}$ satisfies ϕ, written $\mathscr{M} \models \phi$, if and only if the Even player has a winning strategy in $\mathscr{M} \times \phi$.

Definition 6. *(Model-checking parity game)* For any formula ϕ of modal μ, and a model $\mathscr{M}$, define a parity game $\mathscr{M} \times \phi$ with positions (s, ψ) where s is a state of $\mathscr{M}$ and ψ is either a proper subformula of ϕ, or the formula $\bigvee \mathscr{B}$, or the formula $\Diamond \psi$ for any $\rightarrow \mathscr{B}$ and $\psi \in \mathscr{B}$ in ϕ. The initial position is (s_0, ϕ) where s_0 is the root of $\mathscr{M}$. Positions (s, ψ) where ψ is a disjunction or $\Diamond \psi'$ belong to Even while conjunctions and positions $\rightarrow \mathscr{B}$ belong to Odd. Other positions have at most one successor; let them be Even's although the identity of their owner is irrelevant. There are edges from $(s, \psi \vee \psi')$ and $(s, \psi \wedge \psi')$ to both (s, ψ) and (s, ψ'); from $(s, \mu X.\phi)$ and $(s, \nu X.\phi)$ to (s, ϕ); from (s, X) to $(s, \nu X.\psi)$ if X is bound by ν, or $(s, \mu X.\psi)$ if it is bound by μ; finally, from $(s, \rightarrow \mathscr{B})$ to every $(s', \bigvee \mathscr{B})$ where (s, s') is an edge in $\mathscr{M}$, and also to every $(s, \Diamond \psi)$ where $\psi \in \mathscr{B}$ and from $(s, \Diamond \psi)$ to every (s', ψ) where (s, s') is an edge in the model $\mathscr{M}$. Positions $(s, P), (s, \neg P), (s, \top)$ and $(s, \bot)$ have no successors. The parity function assigns an even priority to $(s, \top)$ and also to (s, P) if s satisfies P in $\mathscr{M}$ and to $(s, \neg P)$ if s does not satisfy P in $\mathscr{M}$; otherwise (s, P) and $(s, \neg P)$ receive odd priorities, along with $(s, \bot)$. Fixpoint variables are given positive integer priorities such that ν-bound variables receive even priorities while μ-bound variables receive odd priorities. Furthermore, whenever X has priority i, Y has priority j and $i < j$, X must not appear free in the formula ψ binding Y in $\mu Y.\psi$ or $\nu Y.\psi$. In other words, inner fixpoints receive lower, less significant priorities while outer fixpoint receive high priorities. Other nodes receive the least priority used, 0 or 1.

We now use parity games to define the semantics of L_μ.

Definition 7. *(Satisfaction relation)* A structure $\mathscr{M}$, rooted at s_0 is said to satisfy a formula Ψ of L_μ, written $\mathscr{M} \models \Psi$ if and only if the Even player has a winning strategy from (s_0, Ψ) in $\mathscr{M} \times \Psi$.

Note that the definition of the model-checking parity game requires a priority assignment to fixpoint variables in a formula that satisfies the conditions that ν-variables receive even priorities, μ-variables receive odd priorities and whenever X has priority i, Y has priority j and $i < j$, X must not appear free in the formula ψ binding Y in $\mu Y.\psi$ or $\nu Y.\psi$. For any formula, there are several valid assignments. For example, one could assign a distinct priority to every fixpoint, with the highest priority going to the outermost bound fixpoint and the priorities decreasing the further into the formula a fixpoint is bound. We further restrict a parity assignment to be surjective into an initial fragment of $\mathbb{N}$: if a priority is unused, all greater priorities can be reduced by 2. We define the alternation depth of a formula to be the minimal valid assignment. Although variations of this definition exists, our motivation is to match closely the alternations required in the model checking parity game.

Definition 8. Let a priority assignment be a function $\Omega : Var \rightarrow \{0...n\}$ for some integer n, which is surjective on at least $\{1, ..., n\}$, such that if $\Omega(X) < \Omega(Y)$ then X does not appear free in the formula binding Y and the parity of $\Omega(X)$ is even for ν-bound variables and odd for μ-bound variables. We don't require the priority 0 to be used, but include it in the co-domain for simplicity. In this paper, we take the alternation depth of a formula to be the co-domain of the least priority assignment of a formula. The correspondance with the priorities of the model checking parity game should make it clear that this definition is equivalent to the more typical syntactic ones in the literature, for example in [2]. An alternation free formula is a formula which has both priority assignements with co-domain $\{0, 1\}$ and $\{0, 1, 2\}$ where 0 is not used.

Deciding whether a formula is equivalent to a formula with smaller alternation depth is a long standing open problem.

2.2 Tableau decomposition

Definition 9. *(Tableau)* A tableau $\mathscr{T} = (T, L)$ of a formula Ψ consists of a potentially infinite tree T of which each node n has a label $L(n) \subseteq sf(\Psi)$ where $sf(\Psi)$ is the set of proper subformulas of Ψ. The labelling respects the following tableau rules with the restriction that the modal rule is only applied where no other rule is applicable.

$$\frac{\{\Gamma, \phi, \psi\}}{\{\Gamma, \psi \wedge \phi\}}\,(\wedge) \qquad \frac{\{\Gamma, \phi\} \qquad \{\Gamma, \psi\}}{\{\Gamma, \psi \vee \phi\}}\,(\vee) \qquad \frac{\{\Gamma, \phi\}}{\{\Gamma, \sigma X.\phi\}}\,(\sigma) \text{ with } \sigma \in \{\mu, \nu\}$$

$$\frac{\{\Gamma, \phi\}}{\{\Gamma, X\}}\,(X) \text{ where } X \text{ is a fixpoint variable bound by } \sigma X.\phi, \text{ with } \sigma \in \{\mu, \nu\}$$

$$\frac{\{\psi\} \cup \{\bigvee \mathscr{B} | {\to}\mathscr{B} \in \Gamma, \mathscr{B} \neq \mathscr{B}'\} \text{ for every } {\to}\mathscr{B}' \in \Gamma, \psi \in \mathscr{B}'}{\{\Gamma\}}\,(\to)$$

Note that each branching node is either a choice node, corresponding to a disjunction, or a modal node. Although the rules only contain a binary disjunctive rule, we may write, for the sake conciseness, a sequence of binary choice nodes as a single step. Also note that when a modal rule is applied, all formulas in a label are either modal formulas or literals, that is to say propositional variables and their negations. The latter form the modal node's set of literal and are a semantically important component of the tableau. An inconsistent set of literals is equivalent to $\bot$ and a node with such a set of literals in its label has no successors.

Sequences of subformulas along a path in the tableau are called traces and correspond to plays in the model checking parity game. A μ-trace is a trace winning for the Odd player.

Definition 10. *(μ-trace)* Given an infinite branch in a tableau, that is to say a sequence $n_0 n_1 \ldots$ of nodes starting at the root, where n_{i+1} is a child of n_i, a trace on it is an infinite sequence $f_0 f_1 \ldots$ of formulas satisfying the following: each formula is taken from the label of the corresponding node, $f_i \in L(n_i)$ for all $i \geq 0$; successive formulas f_i and f_{i+1} are identical if f_i is not the formula that the tableau rule from n_i to n_{i+1} acts on; if the tableau rule from n_i to n_{i+1} is a disjunction, conjunction, or fixpoint binding elimination acting on f_i, then f_{i+1} is an immediate subformula of f_i; if the tableau rule from n_i to n_{i+1} is a modality, then f_i has to be a formula ${\to}\mathscr{B}$ and f_{i+1} is either $\bigvee \mathscr{B}$ or a formula $\psi \in \mathscr{B}$; if the tableau rule from n_i to n_{i+1} is a fixpoint regeneration acting on the fixpoint variable f_i, then f_{i+1} is the binding formula for f_i. A trace is a μ-trace if the most significant fixpoint variable that regenerates infinitely often on it is a μ-variable.

Since labels are to be thought of as conjuncts, it is sufficient for an infinite path in a tableau to allow one μ-trace for the infinite path to be winning for the Odd player.

Definition 11. *(Parity of a path)* An infinite path in a tableau is said to be even if there are no μ-traces on it, otherwise it is said to be odd.

Note that the order of applications of the tableau rules is non deterministic so a formula may appear to have more than one tableau. However, tableau equivalence, defined next, only looks at the structure of branching, whether branching nodes are modal or disjunctive, the literals at modal nodes and the parity of infinite paths, so a formula has a unique tableau, up to tableau equivalence. We define tableau cores

to be the semantic elements of the tableau – node types, literals at modal nodes, branching structure and the parity of infinite paths – which do not depend on the syntax of the generating formula. Finally, we define trees with back edges which are finite representations of tableau cores.

Definition 12. *(Tableau core)* A tableau core is $\mathscr{C} = (C, \Omega)$ where C is a potentially infinite but still finitely branching tree of which the nodes are either modal nodes or disjunctive nodes and modal nodes are decorated with a set of literals. Ω is a parity assignment with a finite prefix of $\mathbb{N}$ as co-domain. An infinite path in $\mathscr{C}$ is of the parity of the most significant priority seen infinitely often. $\mathscr{C} = (C, \Omega)$ is a tableau core for $\mathscr{T} = (T, L)$ if once the sequences of disjunctions in $\mathscr{T}$ are collapsed into one non-binary disjunction there is a bijection b between the branching nodes of T and the nodes of C which respects the following: the successor relation in the sense that $b(i)$ is a child of $b(j)$ in C if and only if i is a child of j in T, whether nodes are modal or disjunctive, the literals at modal nodes, and the parity of infinite paths. That is to say, if a path in $\mathscr{T}$ maps to a path in $\mathscr{C}$ then the highest priority seen infinitely often on the path in $\mathscr{C}$ is even if and only if the path in $\mathscr{T}$ has no μ-trace.

Definition 13. *(Tableau equivalence)* Two tableaus $(\mathscr{T}_0, L_0)$ and $(\mathscr{T}_1, L_0)$ are equivalent if their cores are bisimilar with respect to their branching structure, whether nodes are disjunctive or modal, the literals at modal nodes and the parity of infinite branches. Two formulas are tableau equivalent if they generate equivalent tableaus.

Definition 14. *(Tree with back edges)* Tableaus are potentially infinite but regular, so they allow finite representations. A finite representation of a tableau $\mathscr{A} = (A, \Omega)$ is a finite tree with back edges, A which is bisimilar to the core of the tableau. Every node is either a modal node or a disjunctive node and modal nodes are associated with a set of literals. The tree has a priority assignment Ω which assigns priorities to nodes such that the highest priority on an infinite path is of the parity of that path.

To summarise, a tableau $\mathscr{T}$ is a potentially infinite tree labelled with sets of subformulas – it is specific to the formula which labels its root; a tableau core, $\mathscr{C}$ is a potentially infinite object which carries the same semantics but is not specific to one formula; finally, a tree with back edges, called $\mathscr{A}$ because of its resemblance to alternating parity automata, is a finite representation of a tableau core. The next section will present the one-to-one correspondence between disjunctive formulas and trees with back edges.

Theorem 15. *[9] Tableau equivalent formulas are semantically equivalent.*

Note that tableau equivalence is a stricter notion than semantic equivalence; $\psi \vee \neg\psi$ and $\top$ have different tableau for example.

2.3 Disjunctive normal form

Disjunctive form was introduced in [9] as a syntactic restriction on the use of conjunctions. It forces a formula to follow a simple structure of alternating disjunctions and modalities where modalities are qualified with a conjunction of propositions. Such formulas are in many ways well-behaved and easier to manipulate than arbitrary L_μ formulas.

Definition 16. *(Disjunctive formulas)* The set of disjunctive form formulas of (unimodal) L_μ is the smallest set $\mathscr{F}$ satisfying:

- $\bot, \top$, propositional variables and their negations are in $\mathscr{F}$;
- If $\psi \in \mathscr{F}$ and $\phi \in \mathscr{F}$ then $\psi \vee \phi \in \mathscr{F}$;
- If $\mathscr{A}$ is a set of literals and $\mathscr{B} \subseteq \mathscr{F}$ ($\mathscr{B}$ is finite), then $\bigwedge \mathscr{A} \wedge \rightarrow \mathscr{B} \in \mathscr{F}$;

- $\mu X.\psi$ and $\nu X.\psi$ as long as $\psi \in \mathscr{F}$.

Every formula is known to be equivalent to an effectively computable formula in disjunctive form [9]. The transformation into disjunctive form involves taking the formula's tableau decomposition and compressing the node labels into a single subformula. The tricky part is finding a tree with back edges and its priority assignment to represent the tableau finitely, including the parity of infinite paths. The transformation then turns the tree with back edges into a disjunctive formula with alternation depth dependent on the priority assignment. Conversely, a disjunctive formula and its minimal priority assignment induces a tree with back edges representing its tableau. The minimal priority function required to finitely represent a tableau is therefore equivalent to the minimal alternation depth of a disjunctive formula generating the tableau. The following theorem recalls the construction of disjunctive formulas from trees with back edges labelled with priorities from [9] and shows that the alternation depth of the resulting formula stems from the priority assignment of the tree with back edges.

Theorem 17. *Let $\mathscr{A} = (A, \Omega)$ be a tree with back edges that is bisimilar to a core of the tableau $\mathscr{T}$ with priority assignment Ω with co-domain $\{0...q\}$. Then there is a disjunctive formula with alternation depth $\{0...q\}$ which generates a tableau equivalent to $\mathscr{T}$.*

Proof. First of all, we construct $\mathscr{A}' = (A', \Omega')$, bisimilar to $\mathscr{A}$ but with a priority assignment with the following property: on all paths from root to leaf, the priorities of nodes that are the targets of back edges occur in decreasing order. This is straight-forward by looking at the infinite tableau core $\mathscr{A}$ unfolds into, remembering which nodes stem from the same node in $\mathscr{A}$ and their priority assigned by Ω. First consider all branches that see the highest priority q infinitely often and cut them short by creating back edges at nodes of priority q, pointing to the bisimilar ancestor node (also of priority q) that is closest to the root. Then repeat this for each priority in decreasing order, but for each priority $q-1$ treat the ancestor of priority q that back edges point to (if it exists) as the root, so that nodes that have back edges pointing to them end up in decreasing order of priority. Note that every cycle is now dominated by the priority of the first node from the root seen infinitely often.

The disjunctive formula is then obtained by assigning a subformula $f(n)$ to every node of A as follows. If n is a leaf with literals Q, then $f(n) = \bigwedge Q$; if n is a disjunctive node with children n_0 and n_1, then $f(n) = f(n_0) \vee f(n_1)$; if n is the source of a back edge of which the target is m, then $f(n) = X_m$ where X_m is a fixpoint variable; if n is a modal node, then $f(n) = \bigwedge Q \wedge \rightarrow \mathscr{B}$ where Q is the set of literals at n and $\mathscr{B}$ is the set of $f(n_i)$ for n_i children of n; other nodes inherit the formula assigned to their unique child. If n is the target of a back edge, $f(n)$ is obtained as previously detailed but in addition, it binds the fixpoint variable X_n with a ν-binding if n is of even parity and with a μ-binding otherwise.

If r is the root node of A', then $f(r)$ is a disjunctive formula that generates a tableau that is equivalent to $\mathscr{T}$. This should be clear from the fact that the tableau of $f(n)$ consists of the infinite tree generated by A' and the labelling $L(n) = \{f(n)\}$ for all n. Ω' restricted to the target of back edges is a priority assignment for the disjunctive formula $\Psi = f(n)$ since it respects the parity of paths and on each branch the priorities occur in decreasing order. This guarantees that if $\Omega'(X) < \Omega'(Y)$ then X is not free in the formula binding Y.

Therefore Ψ has a tableau that is equivalent to $\mathscr{T}$ and accepts a priority assignment with co-domain $\{0...q\}$. □

Conversely, a disjunctive formula induces a tree with back edges generating its tableau by taking its tableau until each branch reaches a fixpoint variable which is the source of a back edge to its binding formula. The priority assignment of the formula is also a priority assignment for the tree with back-edges. This yields a one-to-one correspondence between trees with back edges and disjunctive formulas.

3 Tableau equivalence preserves alternation depth for disjunctive L_μ

This section argues that all disjunctive formulas generating the same tableau $\mathscr{T}$ have the same alternation depth. The structures used to identify the alternation depth are similar to ones found in [8] to compute the Rabin-Mostowski index of a parity games and the flowers described in [12] to find the Rabin-Mostowski index of non-deterministic automata. Here I show that tableau equivalence preserves these structures and consequently also the alternation depth of disjunctive formulas.

Definition 18 describes a witness showing that the priority assignment Ω of a tree with back edges $\mathscr{A} = (A, \Omega)$ representing $\mathscr{T}$ requires at least q priorities. This witness is preserved by bisimulation with respect to node type, literals and parity of infinite branches. Since all finite representations of a tableau $\mathscr{T}$ are bisimilar with respect to these criteria, they all have the same maximal witness, indicating the least number of priorities $\mathscr{T}$ can be represented with.

Informally, the witness of strictness is a series of cycles of alternating parity where each cycle is contained within the next.

Definition 18. *(q-witness)* A q-witness in a tree with back edges (A, Ω) representing a tableau $\mathscr{T}$ consists of q cycles $c_1 ... c_q$ such that for each $i \leq q$, the cycle c_i is of the parity of i and for all $0 < i < q$, the cycle c_i is a subcycle of c_{i+1}.

Lemma 19. *If a tree with back edges (A, Ω) has a q-witness, then the co-domain of the priority assignment Ω has at least q elements.*

Proof. Given a q-witness $c_1 ... c_q$, for every pair of cycles c_i and c_{i+1}, since they are of different parity and c_i is contained in c_{i+1}, the dominant priority on c_{i+1} must be strictly larger than the dominant priority on c_i. Therefore there must be at least q priorities in the cycle c_q which contains all the other cycles of the witness. □

Lemma 20. *If a tree with back edges $\mathscr{A}$ representing a tableu $\mathscr{T}$ does not have a q-witness, then there is an tree with back edges $\mathscr{A}'$ which also represents $\mathscr{T}$ but has a priority assignment with fewer priorities.*

Proof. Assume a tree with back edges $\mathscr{A} = (A, \Omega)$ representing $\mathscr{T}$ with a priority assignment with co-domain $\{0...q\}$ does not have a q witness. Let S_q be the set of nodes of priority q. Let S_{i-1} for $1 < i \leq q$ be the set of nodes of priority $i-1$ which appear as the second highest priority in a cycle where all the nodes of highest priority are in S_i, and as the nodes of highest priority in some cycle. Note that if S_1 was non-empty, then there would be a q-witness, so S_1 and consequently S_0 must be empty. Then define a new priority function as follows: the new priority function Ω' is as Ω, except for nodes in any S_i – these receive the priority $i-2$ instead of the priority i. Since S_1 and S_0 are empty, this is possible whilst keeping all priorities positive. Ω' with co-domain $\{0...q-1\}$ preserves the parity of infinite branches since there are no cycles in which the priority of all dominant nodes is decreased more than the priority of all sub-dominant nodes and each node retains the same parity. Therefore, if a finite representation of $\mathscr{T}$ does not have a q-witness, then there is a finite representation $\mathscr{A}' = (A, \Omega')$ with a smaller priority assignment. □

Lemma 21. *All tableau equivalent trees with back edges have the same q-witnesses: for all q, either all or none of the trees with back edges representing a same tableau $\mathscr{T}$ have a q-witness.*

Proof. First we recall that if $\mathscr{A}$ is the finite representation of $\mathscr{T}$ induced by a disjunctive formula Ψ then the tableau of $\mathscr{T}$ is an infinite tree bisimilar to $\mathscr{A}$ with respect to node type, literals and parity of

infinite branches. Hence any finite representation of $\mathscr{T}$ is bisimilar to $\mathscr{A}$. It then suffices to show that q-witnesses are preserved under bisimulation. This is straight-forward: let $\mathscr{A}'$ be bisimilar to a finite tree with back edges $\mathscr{A}$ with respect to node type, literals at modal nodes and the parity of infinite paths. Then infinite paths in $\mathscr{A}$ are bisimilar to infinite paths in $\mathscr{A}'$. Since both $\mathscr{A}$ and $\mathscr{A}'$ are finite, an infinite path stemming from a cycle in $\mathscr{A}$ is bisimilar to a cycle in $\mathscr{A}'$. A q-witness contains at least one node which lies on all the cycles of the witness. If $\mathscr{A}$ has q cycles, call the node on all of its cycles n and consider (one of) the deepest node(s) n' in $\mathscr{A}'$ bisimilar to n. That is to say, choose n' such that if another node bisimilar to n' is reachable from n', it must be an ancestor of n'. Since n' is bisimilar to n, there must be a cycle c'_i bisimilar to each c_i reachable from n'. Since n' is maximally deep, it is contained in each of these cycles c'_i. Then, a q-witness can be reconstructed in $\mathscr{A}'$ by taking the cycle c'_1, and then for each $i > 0$ the cycle consisting of all $c'_j, j \leq i$. Since all c'_i cycles have n' in common, there is a cycle combining $c'_j, j \leq i$ for any i. Since bisimulation respects the parity of cycles, this yields a q-witness in $\mathscr{A}'$. □

Theorem 22. *All disjunctive formulas with tableau $\mathscr{T}$ have the same alternation depth.*

Proof. All trees with back edges representing the same tableau $\mathscr{T}$ have the same maximal witness, from the previous lemma, so from Lemma 20 they accept a minimal priority function with domain $\{0...q\}$. Since a disjunctive formula induces a tree with back edges with a minimal priority function corresponding to the formula's alternation depth, any two disjunctive formulas that are tableau equivalent must have the same alternation depth. □

This concludes the proof that tableau equivalence preserves alternation depth on disjunctive formulas. The restriction to disjunctive formulas is crucial: as the next section shows, in the general case tableau equivalent formulas may have vastly different alternation depths.

4 Disjunctive form does not preserve alternation depth

Every formula has a tableau which allows it to be turned into a semantically equivalent disjunctive formula. This section studies the relationship between a formula's alternation depth and the alternation depth of its tableau equivalent disjunctive form. As the previous section shows, any two disjunctive formulas with the same tableau have the same alternation depth; therefore comparing a non-disjunctive formula to any tableau equivalent disjunctive formula will do.

The first subsection demonstrates that not only does disjunctive form not preserve alternation depth, but also that there is no hope for bounding the alternation depth of disjunctive formulas with respect to their semantic alternation depth: for any n there are one alternation formulas which are tableau equivalent to n alternation disjunctive formulas. In other words, the alternation depth of a L_μ formula, when transformed into disjunctive form, can be arbitrarily large. Conversely, as shown in the second subsection, formulas of arbitrarily large alternation depth can be tableau equivalent to a disjunctive formula without alternations. Hence the alternation depth of tableau equivalent formulas are only related within the disjunctive fragment.

4.1 Disjunctive formulas with large alternation depth

While the main theorem is proved by Example 27, the Examples 23 and 25 leading up to it should give the interested reader some intuition about the mechanics which lead the tableau of a formula to have higher alternation depth than one might expect.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{Y,X \quad \mathbf{W}}}{\to\{Y\},\bar{A},\to\{X\} \quad \bar{\mathbf{A}},\to\{\mathbf{W}\}}}{\to\{Y\},\bar{A}\wedge\to\{X\} \quad \bar{\mathbf{A}}\wedge\to\{\mathbf{W}\}}}{* \quad \to\{Y\},(\bar{A}\wedge\to\{X\})\vee A \quad (\bar{\mathbf{A}}\wedge\to\{\mathbf{W}\})\vee(\mathbf{A}\wedge\to\{\mathbf{Z}\})} \qquad \dfrac{\dfrac{\dfrac{*}{Y \quad \mathbf{Z}}}{\to\{Y\},A \quad \mathbf{A},\to\{\mathbf{Z}\}}}{\to\{Y\},A \quad \mathbf{A}\wedge\to\{\mathbf{Z}\}}}{\nu Y.\to\{Y\}\wedge\mu X.(\bar{A}\wedge\to\{X\})\vee A \quad \nu\mathbf{Z}.\mu\mathbf{W}.(\bar{\mathbf{A}}\wedge\to\{\mathbf{W}\})\vee(\mathbf{A}\wedge\to\{\mathbf{Z}\})}$$

Figure 4.1: Tableaus for $\nu Y.\to\{Y\}\wedge\mu X.(\bar{A}\wedge\to\{X\})\vee A$ and $\nu\mathbf{Z}.\mu\mathbf{W}.(\bar{\mathbf{A}}\wedge\to\{\mathbf{W}\})\vee(\mathbf{A}\wedge\to\{\mathbf{Z}\})$

Example 23. The first example is a rather simple one: a disjunctive formula with one alternation that can be expressed in non-disjunctive form without any alternations. The disjunctive formula $\nu X.\mu Y.(A\wedge\to\{X\})\vee(\bar{A}\wedge\to\{Y\})$ signifies that all paths are infinite and A occurs infinitely often on all paths. Compare it to the formula $\nu X.\to\{X\}\wedge\mu Y.(\bar{A}\wedge\to\{Y\})\vee A$ which is alternation free.

The tableaus of both these formulas are shown side by side in Figure 4.1. Both branches regenerate into either exactly the ancestral node marked * or a node that reaches a node identical to the one marked * in a single non branching step.
The cores of the two tableaus, that is to say their branching nodes, are clearly isomorphic with respect to the node type and branching structure. Furthermore, for both formulas, there is μ-trace on any path that only goes through the left hand branch infinitely often. There is no μ trace on *any* path that goes through the right hand path infinitely often, for either formula. As a result, both tableaus agree on the parity of infinite branches. The two formulas are tableau equivalent and therefore also semantically equivalent.

Remark 24. Observe that there is nothing obviously inefficient about how the disjunctive formula handles alternations. Indeed, simply inverting the order of the fixpoints yields a formulas which can not be expressed without an alternation: $\mu X.\nu Y.A\wedge\to\{X\}\vee\bar{A}\wedge\to\{Y\}$.

While the above example proves that disjunctive form does not preserve alternation, it must be noted that the alternating parity automata corresponding to these formulas require in both cases two priorities, although only one requires an alternation. The next example shows formulas in which the number of priorities is not preserved either.

Example 25. This example and the following ones will be built on one-alternation formulas consisting of single μ/ν alternations embedded in one another without interfering with each other, i.e. all free variables within the inner formula ϕ_1 are bound by the inner fixpoint bindings. This means that the formula accepts a priority assignment with co-domain $\{0,1\}$. Without further ado, consider the formula in question:

$$\alpha = \mu X_0.\nu Y_0.(A\wedge\to\{X_0\})\vee(B\wedge\to\{Y_0\}\wedge\mu X_1.\nu Y_1.(C\wedge\to\{X_1\})\vee(D\wedge\to\{Y_1\})\vee E$$

The following Lemma shows it to be equivalent to a formula which requires a priority assignment with co-domain $\{0...3\}$.

Lemma 26. *The formula α is tableau equivalent to a disjunctive formula which requires a parity assignment with co-domain $\{0...3\}$:*

$$\begin{aligned}\beta = \mu X_0.\nu Y_0.\mu X_1.\nu Y_1.&(A\wedge C\wedge\to\{X_0\})\vee(A\wedge D\wedge\to\{X_0\})\vee(A\wedge E\wedge\to\{X_0\})\\ &\vee(B\wedge E\wedge\to\{Y_0\})\vee(B\wedge C\wedge\to\{X_1\})\vee(B\wedge D\wedge\to\{Y_1\})\end{aligned} \tag{4.1}$$

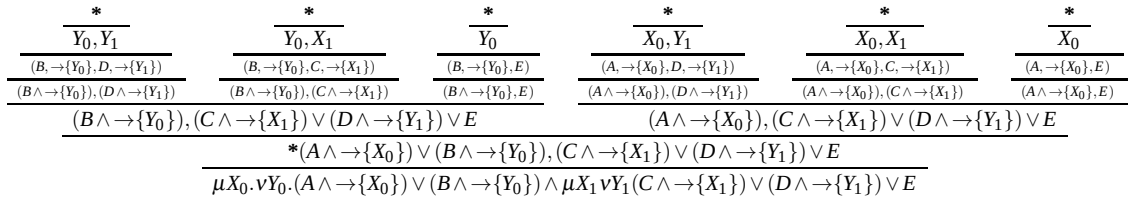


Figure 4.2: Tableau for α

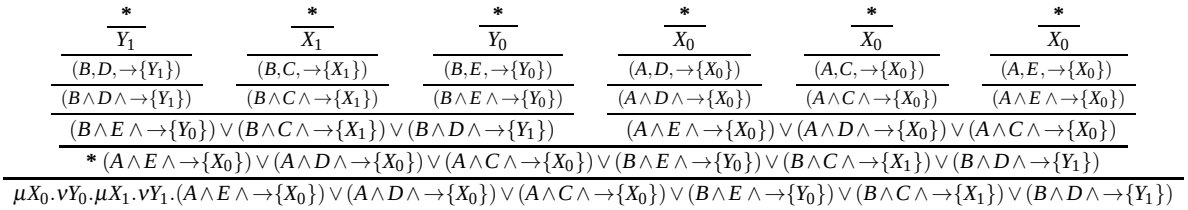


Figure 4.3: Tableau for β

Proof. The tableaus for both formulas are written out in Figures 4.2 and 4.3. The two tableaus are isomorphic with respect to branching structure, node type and the literals at modal nodes. To prove their equivalence, it is therefore sufficient to argue that this isomorphism also preserves the parity of infinite branches, that is to say that there is a μ-trace in an infinite path of one if and only if there is a μ-trace in the corresponding infinite path of the other.

To do so, we look, case by case, at the combinations of branches that a path can see infinitely often and check which have a μ trace in each tableau. First argue that the three right-most branches in both tableaus are such that any path that sees them infinitely often has a μ-trace. This is witnessed in both cases by the least fixpoint variable X_0 which will dominate any trace it appears on and appears on a trace on all paths going through one of these branches infinitely often. So, in both tableaus, any path going through one of the right-most branches infinitely often is of odd parity. Now consider the branch that ends in Y_0 before regenerating to the node marked * in both tableaus. All traces on paths that go infinitely often through this branch will see Y_0 regenerate infinitely often. Therefore in both tableaus, a path going through this branch infinitely has a μ trace if and only if it also goes through one of the three rightmost branches infinitely often. Now consider the fifth branch from the right, the branch that regenerates Y_0, X_1 in one case and just X_1 in the other. In both tableaus, a path that goes through this branch infinitely often will have a μ trace unless it goes through the Y_0 branch infinitely often and doesn't go through one of the three right-most branches infinitely often. Finally, in both tableaus, a branch that only sees the left-most branch infinitely often is of even parity since such a path does not admit any μ-traces. However, if a path sees this branch and some other branches infinitely often, its parity is determined by one of the previously analysed cases. Since we have analysed all the infinite paths on these tableaus and concluded that in each case the parity of a path is the same in both tableaus, this concludes the proof that the two tableaus are equivalent. □

The above example yields a disjunctive formula of alternation depth $\{0...3\}$ which semantically only requires alternation depth $\{0,1\}$. This proves that disjunctive form does not preserve the number of priorities the model checking game of a formula requires.

The next step is to generalise the construction of Example 25 to arbitrarily many alternations to prove that there is no bound on the number of alternations of a disjunctive formula tableau equivalent to a non-disjunctive formula of n alternations. To do so, we will first define the one-alternation formulas α_n inductively, based on the formula of Example 25. We then argue that the tableau of α_n admits a $(2n+1)$-witness, proving that α_n is not tableau equivalent to any disjunctive formula of less than $2n+1$ alternations. Due to the argument pertaining to traces in increasingly large tableaus, its details are, inevitably, quite involved. However, the mechanics of the tableaus of α_n are not difficult; writing down the tableau of α_2 and working out its disjunctive form should suffice to gain an intuition of the proof to follow.

Example 27. In order to define α_n for any n define:

$$\begin{aligned} a_1 &= \mu X_1.\nu Y_1.((A_1 \wedge \to\{X_1\}) \vee (B_1 \wedge \to\{Y_1\}) \vee E_1) \wedge \\ &\quad \mu X_0.\nu Y_0.(A_0 \wedge \to\{X_0\}) \vee (B_0 \wedge \to\{Y_0\}) \vee E_0 \\ a_{i+1} &= \mu X_{i+1}.\nu Y_{i+1}.((A_{i+1} \wedge \to\{X_{i+1}\}) \vee (B_{i+1} \wedge \to\{Y_{i+1}\}) \vee E_{i+1}) \wedge a_i \end{aligned} \tag{4.2}$$

Then, define:

$$\alpha_n = \mu X_n.\nu Y_n.((A_n \wedge \to\{X_n\}) \vee (B_n \wedge \to\{Y_n\})) \wedge a_{n-1}$$

In other words, the formula consists of nested clauses $\mu X_i.\nu Y_i.((A_i \wedge \to\{X_i\}) \vee (B_i \wedge \to\{Y_i\}) \vee E_i)$ connected by conjunctions where the outmost clause does not have a $\vee E$.

As the formula grows, its tableau becomes unwieldy, but its structure remains constant: it is just as the tableau of α with more branches. Figure 4.2 can be used as reference.

The tableau of any α_n follows this structure:

- The first choice node $\{(A_n \wedge \to\{X_n\} \vee B_n \wedge \to\{Y_n\}), ..., (A_0 \wedge \to\{X_0\} \vee B_0 \wedge \to\{Y_0\} \vee E_0)\}$ branches into 2×3^n modal nodes – ignoring the modalities attached to each literals for a moment, this is the decomposition of $(A_n \vee B_n) \wedge (A_{n-1} \vee B_{n-1} \vee E_{n-1}) ... \wedge (A_0 \vee B_0 \vee E_0)$ into one large disjunction.
- Each choice leads to a modal node with some choice of propositional variables consisting of one of A_n and B_n and then for every $i < n$ one of A_i, B_i or E_i.
- These modal nodes have a single successor each, consisting of a set of fixpoint variables. In every case, one of these is Y_n or X_n and there is only ever at most one fixpoint variable out of $\{X_i, Y_i\}$ for each i. These nodes will be referred to as regeneration nodes. When a regeneration node does not contain X_i nor Y_i for some i, this corresponds to E_i having been chosen rather than A_i or B_i.
- Nodes consisting of a set of fixpoint variables all regenerate, give or take a couple of non-branching steps, into the same choice node, identical to the ancestral choice node labelled:

$$\{(A_n \wedge \to\{X_n\} \vee B_n \wedge \to\{Y_n\}), ..., (A_0 \wedge \to\{X_0\} \vee B_0 \wedge \to\{Y_0\} \vee E_0)\}$$

- An infinite trace in this tableau sees infinitely often only fixpoint variables Y_i and/or X_i for some i. As a consequence if a path goes infinitely often through a regeneration node which does not contain X_i or Y_i, then there is no trace that sees X_i infinitely often on that path.

Lemma 28. *The formula α_n is tableau equivalent only to disjunctive formulas which require a priority assignment with $2n+1$ priorities.*

Proof. Using the above observations, we will show that the tableau for this formula requires at least $2n+1$ alternating fixpoints. We describe a priority assignment to a subset of the nodes of the tableau of α_n such that on the paths within this subset, a path is even if and only if the most significant priority seen infinitely often is even. We then argue that this subset constitutes a $2n+1$-witness.

Consider the paths of the tableau which only contain the following regeneration nodes:

- For all i, the nodes regenerating exactly $Y_nY_{n-1}...Y_i$, and
- For all i the nodes regenerating exactly $Y_n...Y_{i+1}X_iY_{i-1}...Y_0$.

For each i, assign priority $2i$ to the node regenerating $Y_n...Y_i$ and $2i+1$ to the node regenerating $Y_n...Y_{i+1},X_i,Y_{i-1},...Y_0$. We now prove that this priority assignment is such that a path within this subtableau is even if and only if the highest priority seen infinitely often is even.

First consider the nodes $Y_n...Y_i$, which have been assigned even priority. A path that sees such a node infinitely often can only have a μ-trace if it sees a node regerating some X_j, $j > i$ infinitely often. Such a node would have an odd priority greater than $Y_n...Y_i$. Therefore, if the most significant priority seen infinitely often is even, the path has no μ trace. Conversely, if a path sees $Y_n...X_i...Y_0$ infinitely often and no $Y_n...Y_j$ where $j > i$ infinitely often, then there is a trace which only regenerated X_i and Y_i infinitely often. This is a μ trace since X_i is more significant than Y_i. This priority assignment therefore describes the parity of infinite paths on this subset of paths of $\mathscr{T}$.

Any assignment of priorities onto $\mathscr{T}$ should, on this subset of paths, agree in parity with the above priority assignment. However, in any tree with back edges generating this tableau, this subset of paths constitutes a $2n+1$ witness: c_0 is a cycle that only sees $Y_n...Y_0$, c_1 contains c_0 and also sees $Y_n...X_1Y_0$ infinitely often and for all $i > 1$, the cycle c_{2i} is one containing c_{2i-1} and $Y_n...Y_i$ while c_{2i+1} is one containing c_{2i} and $Y_n...X_i...Y_0$. Each cycle c_j is dominated by the priority j, making $c_0,...,c_{2i+1}$ a $2i+1$-witness. Thus, using Theorem 22 any disjunctive formula with tableau $\mathscr{T}$ must require at least $2n+1$ priorities. □

This concludes the proof that for arbitrary n, there are one-alternation L_μ formulas which are tableau equivalent to disjunctive formulas with n alternations.

4.2 Disjunctive formulas with small alternation depth

The previous section showed that transforming a formula into disjunctive form can increase its alternation depth. The converse is much easier to show: there are very simple formulas for which the transformation into disjunctive form eliminates all alternations.

Lemma 29. *For any formula ψ, the formula $(\mu X.\rightarrow\{X\}\vee\rightarrow\bot)\wedge\psi$ is tableau equivalent to a disjunctive formula without ν-operators.*

Proof. The semantics of $(\mu X.\rightarrow\{X\}\vee\rightarrow\bot)\wedge\psi$ are that a structure must not have infinite paths and ψ must hold. Consider $\mathscr{T}$, the tableau for $(\mu X.\rightarrow\{X\}\vee\rightarrow\bot)\wedge\psi$. It is easy to see that every modal node will either contain $\rightarrow\{X\}$ or $\rightarrow\bot$. The latter case terminates that branch of the tableau, while the former will populate every successor node with X which will then regenerate into $(\rightarrow\{X\}\vee\rightarrow\bot)$. As a result, all infinite paths have a μ trace; there are no even infinite paths. Any disjunctive formula generating $\mathscr{T}$ will therefore only require the μ operator. □

Taking ψ to be a formula of arbitrarily high alternation depth, $(\mu X.\rightarrow\{X\}\vee\rightarrow\bot)\wedge\psi$ shows that the transformation into disjunctive form can reduce the alternation depth an arbitrarily large amount. Together with the previous section, this concludes the argument that there are no bounds on the difference in alternation depth of tableau equivalent formulas.

5 Discussion

To summarise, we have studied how tableau decomposition and the transformation into disjunctive form affects the alternation depth of a formula. The first observation is that within the confines of the disjunctive fragment of L_μ, alternation depth is very well-behaved with respect to tableau equivalence: any two tableau equivalent disjunctive formulas have the same alternation depth. However, the story is quite different for L_μ without the restriction to disjunctive form: the alternation depth of a L_μ formula can not be used to predict any bounds on the alternation depth of tableau equivalent disjunctive formulas and vice versa.

Part of the significance of this result are the implications for our understanding of the alternation hierarchy.

The formulas in Section 4 illutrate some of the different types of accidental complexity which any procedure for deciding the alternation hierarchy would need to somehow overcome. The formula $(\mu X.{\to}\{X\} \vee \bot) \wedge \psi$, from Lemma 29 which is semantically a ν-free formula for any ψ is an example of a type of accidental complexity which the tableau decomposition eliminates. However, the formula in Example 23 illustrate a more subtle form of accidental complexity that is immune to disjunctive form: $\nu X.\mu Y.(A \wedge {\to}\{X\}) \vee (\overline{A} \wedge {\to}\{Y\})$ is semantically alternation free while the syntactically almost identical formula $\mu X.\nu Y.(A \wedge {\to}\{X\}) \vee (\overline{A} \wedge {\to}\{Y\})$ is not. These formulas pinpoint a very specific challenge facing algorithms that try to reduce the alternation depth of formulas; as such, they are valuable case studies for those seeking to understand the L_μ alternation hierarchy.

Finally, we showed that the following is decidable: for any L_μ formula, the least alternation depth of a tableau equivalent disjunctive formula is decidable. This raises the question of whether the same is true if we lift the restriction to disjunctive form, but keep the restriction to tableau equivalence: for a L_μ formula, is the least alternation depth of any tableau equivalent formula decidable? Tableau equivalence is a stricter equivalence to semantic equivalence, so this problem is likely to be easier than deciding the alternation hierarchy with respect to semantic equivalence but it would still be a considerable step towards understanding accidental complexity in L_μ.

Acknowledgements I thank the anonymous reviewers for their thoughtful comments, which have helped improve the presentation of this paper and relate this work to similar results for other automata.

6 Bibliography

References

[1] J.C. Bradfield (1996): *The modal mu-calculus alternation hierarchy is strict*. In U. Montanari & V. Sassone, editors: *CONCUR '96: Concurrency Theory, Lecture Notes in Computer Science* 1119, Springer Berlin Heidelberg, pp. 233–246, doi:10.1007/3-540-61604-7_58.

[2] J.C. Bradfield & C. Stirling (2007): *Modal mu-calculi*. *Handbook of modal logic* 3, pp. 721–756, doi:10.1016/S1570-2464(07)80015-2.

[3] O. Carton & R. Maceiras (1999): *Computing the Rabin index of a parity automaton*. *RAIRO - Theoretical Informatics and Applications - Informatique Thorique et Applications* 33(6), pp. 495–505, doi:10.1051/ita:1999129. Available at http://eudml.org/doc/92617.

[4] T. Colcombet & C. Löding (2008): *The non-deterministic Mostowski hierarchy and distance-parity automata*. In: *Automata, languages and programming*, Springer, pp. 398–409, doi:10.1007/978-3-540-70583-3_33.

[5] G. D'Agostino & M. Hollenberg (2000): *Logical Questions Concerning The mu-Calculus: Interpolation, Lyndon and Los-Tarski*. *J. Symb. Log.* 65(1), pp. 310–332, doi:10.2307/2586539.

[6] E. A. Emerson & C. S. Jutla (1991): *Tree automata, mu-calculus and determinacy*. In: *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, IEEE, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[7] A. Facchini, F. Murlak & M. Skrzypczak (2013): *Rabin-Mostowski index problem: a step beyond deterministic automata*. In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, pp. 499–508, doi:10.1109/LICS.2013.56.

[8] M. Huth, J. Kuo & N. Piterman (2012): *The Rabin Index of Parity Games*. In K. Eder, J. Loureno & O. Shehory, editors: *Hardware and Software: Verification and Testing, Lecture Notes in Computer Science* 7261, Springer Berlin Heidelberg, pp. 259–260, doi:10.1007/978-3-642-34188-5_25.

[9] D. Janin & I. Walukiewicz (1995): *Automata for the modal μ-calculus and related results*. In: *Proc. MFCS '95 LNCS 969*, pp. 552–562, doi:10.1007/3-540-60246-1_160.

[10] M. Jurdziński (2000): *Small progress measures for solving parity games*. In: *STACS 2000*, Springer, pp. 290–301, doi:10.1007/3-540-46541-3_24.

[11] M.K. Lehtinen & S. Quickert (2015): *Deciding the first levels of the modal μ alternation hierarchy by formula construction*. In: *(forthcoming) Proc. CSL '15*.

[12] A. Niwiski & I. Walukiewicz (2005): *Deciding Nondeterministic Hierarchy of Deterministic Tree Automata*. *Electronic Notes in Theoretical Computer Science* 123, pp. 195 – 208, doi:10.1016/j.entcs.2004.05.015. Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC 2004).

[13] D. Niwiski & I. Walukiewicz (1998): *Relating hierarchies of word and tree automata*. In M. Morvan, C. Meinel & D. Krob, editors: *STACS 98, Lecture Notes in Computer Science* 1373, Springer Berlin Heidelberg, pp. 320–331, doi:10.1007/BFb0028571.

[14] I. Walukiewicz (2000): *Completeness of Kozen's axiomatisation of the propositional μ-calculus*. *Information and Computation* 157(1), pp. 142–182, doi:10.1006/inco.1999.2836.

A Type-Directed Negation Elimination

Etienne Lozes
LSV, ENS Cachan & CNRS
lozes@lsv.ens-cachan.fr

In the modal μ-calculus, a formula is well-formed if each recursive variable occurs underneath an even number of negations. By means of De Morgan's laws, it is easy to transform any well-formed formula φ into an equivalent formula without negations – the negation normal form of φ. Moreover, if φ is of size n, the negation normal form of φ is of the same size $\mathcal{O}(n)$. The full modal μ-calculus and the negation normal form fragment are thus equally expressive and concise.

In this paper we extend this result to the higher-order modal fixed point logic (HFL), an extension of the modal μ-calculus with higher-order recursive predicate transformers. We present a procedure that converts a formula of size n into an equivalent formula without negations of size $\mathcal{O}(n^2)$ in the worst case and $\mathcal{O}(n)$ when the number of variables of the formula is fixed.

1 Introduction

Negation normal forms are commonplace in many logical formalisms. To quote only two examples, in first-order logic, negation normal form is required by Skolemization, a procedure that distinguishes between existential and universal quantifiers; in the modal μ-calculus, the negation normal form ensures the existence of the fixed points. More generally, the negation normal form helps identifying the polarities [15] of the subformulas of a given formula; for instance, in the modal μ-calculus, a formula in negation normal form syntactically describes the schema of a parity game.

Converting a formula in a formula without negations – or with negations at the atoms only – is usually easy. By means of De Morgan's laws, negations can be "pushed to the leaves" of the formula. For the modal μ-calculus without propositional variables, this process completely eliminates negations, because well-formed formulas are formulas where recursive variables occur underneath an even number of negations. Moreover, in the modal μ-calculus, if φ is of size n, the negation normal form of φ is of the same size $\mathcal{O}(n)$.

The higher-order fixed point modal logic (HFL) [20] is the higher-order extension of the modal μ-calculus. In HFL, formulas denote either predicates, or (higher-order) predicate transformers, each being possibly defined recursively as (higher-order) fixed points. Since HFL was introduced, it was never suggested that negation could be eliminated from the logic. On the contrary, Viswananthan and Viswanathan [20] motivated HFL with an example expressing a form of rely guarantee that uses negation, and they strove to make sure that HFL formulas are correctly restricted so that fixed points always exist. Negation normal forms in HFL would however be interesting: they would simplify the design of two-player games for HFL model-checking [3], they could help defining a local model-checking algorithms for HFL, they might help to define the alternation depth of a HFL formula, etc.

We show that HFL actually admits negation elimination, and that like for the modal μ-calculus, every HFL formula can be converted into a formula in negation normal form. The negation elimination procedure is more involved due to higher-orderness. As a witness of this increased complexity, our negation elimination procedure has a worst-case quadratic blow-up in the size of the formula, whereas for the μ-calculus the negation normal form is of linear size in the original formula.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 132–142, doi:10.4204/EPTCS.191.12

Related Work Other examples of higher-order recursive objects are the higher-order pushdown automata [17, 4], or the higher-order recursion schemes (HORS) [6, 12, 5, 18]. Whereas the decidability of HFL model-checking against finite transition systems is rather simple, it took more time to understand the decidability of HORS model-checking against the ordinary (order 0) modal μ-calculus. This situation actually benefited to HORS: the intense research on HORS produced several optimized algorithms and implementations of HORS model-checking [2, 9, 19], whereas HFL model-checking remains a rather theoretical and unexplored topic. HORS can be thought as recursive formulas with no boolean connectives and least fixed points everywhere. On the opposite, HFL allows any kinds of boolean connectives, and in particular a form of "higher-order alternation".

Outline We recall the definition of HFL and all useful background about it in Section 2. In Section 3, we sketch the ideas driving our negation elimination and introduce the notion of monotonization, a correspondence between arbitrary functions and monotone ones that is at the core of our negation elimination procedure. We formally define the negation elimination procedure in Section 4, and make some concluding remarks in Section 5.

2 The Higher-Order Modal Fixed Point Logic

We assume an infinite set $\mathsf{Var} = \{X, Y, Z, \dots\}$ of variables, and a finite set $\Sigma = \{a, b, \dots\}$ of labels. Formulas φ, ψ, of the Higher-Order Modal Fixed Point Logic (HFL) are defined by the following grammar

$$\varphi, \psi \quad ::= \top \mid \varphi \vee \psi \mid \neg\varphi \mid \langle a \rangle \varphi \mid X \mid \lambda X^{\tau,\nu}.\ \varphi \mid \varphi\ \psi \mid \mu X^{\tau}.\ \varphi$$

where a type τ is either the ground type Prop or an arrow type $\sigma^{\nu} \to \tau$, and the *variance* ν is either $+$ (monotone), or $-$ (antitone), or 0 (unrestricted). For instance, $\tau_1 = (\mathsf{Prop}^{-} \to \mathsf{Prop})^{+} \to (\mathsf{Prop}^{0} \to \mathsf{Prop})$ is a type, and $\varphi_1 = \lambda F^{\mathsf{Prop}^{-} \to \mathsf{Prop},+}.\ \lambda Y^{\mathsf{Prop},0}.\ \mu Z^{\mathsf{Prop}}.\ (F\ \neg Y) \vee \langle a \rangle (Z \vee \neg Y)$ is a formula. The sets $\mathsf{fv}(\varphi)$ and $\mathsf{bv}(\varphi)$ of free and bound variables of φ are defined as expected: $\mathsf{fv}(X) = \{X\}$, $\mathsf{bv}(X) = \emptyset$, $\mathsf{fv}(\lambda X.\ \varphi) = \mathsf{fv}(\mu X.\ \varphi) = \mathsf{fv}(\varphi) \setminus \{X\}$, $\mathsf{bv}(\lambda X.\ \varphi) = \mathsf{bv}(\mu X.\ \varphi) = \mathsf{bv}(\varphi) \cup \{X\}$, etc. A formula is *closed* if $\mathsf{fv}(\varphi) = \emptyset$. For simplicity, we restrict our attention to formulas φ *without variable masking*, i.e. such that for every subformula $\lambda X.\ \psi$ (resp. $\mu X.\ \psi$), it holds that $X \notin \mathsf{bv}(\psi)$.

Another example is the formula $\varphi_2 = (\lambda F^{\mathsf{Prop}^{-} \to \mathsf{Prop},+}.\ \mu X^{\mathsf{Prop}}.\ F\ X)\ (\lambda Y^{\mathsf{Prop},-}.\ \neg Y)$. This formula can be β-reduced to the modal μ-calculus formula $\varphi_2' = \mu X^{\mathsf{Prop}}.\ \neg X$, which does not have a fixed point semantics. Avoiding ill-formed HFL formulas such as φ_2 cannot just rely on counting the number of negations between μX and the occurence of X, it should also take into account function applications and the context of a subformula.

A type judgement is a tuple $\Gamma \vdash \varphi : \tau$, where Γ is a set of assumptions of the form $X^{\nu} : \tau$. The typing environment $\neg\Gamma$ is the one in which every assumption $X^{\nu} : \tau$ is replaced with $X^{-\nu} : \tau$, where $-+ = -$, $-- = +$, and $-0 = 0$. A formula φ is well-typed and has type τ if the type judgement $\vdash \varphi : \tau$ is derivable from the rules defined in Fig. 1. Intuitively, the type judgement $X_1^{\nu_1} : \tau_1, \dots, X_n^{\nu_n} : \tau_n \vdash \varphi : \tau$ is derivable if asssuming that X_i has type τ_i, it may be infered that φ has type τ and that φ, viewed as a function of X_i, has variance ν_i. For instance, $\vdash \varphi_1 : \tau_1$, where φ_1 and τ_1 are the formula and the type we defined above, but φ_2 cannot be typed, even with different type annotations.

Proposition 1 *[20] If $\Gamma \vdash \varphi : \tau$ and $\Gamma \vdash \varphi : \tau'$ are derivable, then $\tau = \tau'$, and the two derivations coincide.*

$$\frac{}{\Gamma \vdash \top : \mathsf{Prop}} \qquad \frac{\Gamma \vdash \varphi : \tau \qquad \Gamma \vdash \psi : \tau}{\Gamma \vdash \varphi \vee \psi : \tau} \qquad \frac{\neg\Gamma \vdash \varphi : \tau}{\Gamma \vdash \neg\varphi : \tau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Prop}}{\Gamma \vdash \langle a \rangle \varphi : \mathsf{Prop}} \qquad \frac{v \in \{+, 0\}}{\Gamma,\, X^v : \tau \vdash X : \tau}$$

$$\frac{\Gamma, X^v : \sigma \vdash \varphi : \tau}{\Gamma \vdash \lambda X^{v,\sigma}.\, \varphi : \sigma^v \to \tau} \qquad \frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu X^\tau.\, \varphi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^+ \to \tau \qquad \Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

$$\frac{\Gamma \vdash \varphi : \sigma^- \to \tau \qquad \neg\Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^0 \to \tau \qquad \Gamma \vdash \psi : \sigma \qquad \neg\Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

Figure 1: The type system of HFL.

If φ is a well-typed closed formula and ψ is a subformula of φ, we write $\mathsf{type}(\psi/\varphi)$ for the type of ψ in (the type derivation of) φ.

A labeled transition system (LTS) is a tuple $\mathscr{T} = (S, \delta)$ where S is a set of states and $\delta \subseteq S \times \Sigma \times S$ is a transition relation. For every type τ and every LTS $\mathscr{T} = (S, \delta)$, the complete Boolean ring $\mathscr{T}[\![\tau]\!]$ of interpretations of closed formulas of type τ is defined by induction on τ: $\mathscr{T}[\![\mathsf{Prop}]\!] = 2^S$, and $\mathscr{T}[\![\sigma^v \to \tau]\!]$ is the complete Boolean ring of all total functions $f : \mathscr{T}[\![\sigma]\!] \to \mathscr{T}[\![\tau]\!]$ that have variance v, where all Boolean operations on functions are understood pointwise. Note that since $\mathscr{T}[\![\tau]\!]$ is a complete Boolean ring, it is also a complete lattice, and any monotone function $f : \mathscr{T}[\![\tau]\!] \to \mathscr{T}[\![\tau]\!]$ admits a unique least fixed point.

A $\mathscr{T}$-valuation ρ is a function that sends every variable of type τ to some element of $\mathscr{T}[\![\tau]\!]$. More precisely, we say that ρ is well-typed according to some typing environment Γ, which we write $\rho \models \Gamma$, if $\rho(X) \in \mathscr{T}[\![\tau]\!]$ for every $X^v : \tau$ in Γ. The semantics $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!]$ of a derivable typing judgement is a function that associates to every $\rho \models \Gamma$ an interpretation $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!](\rho)$ in $\mathscr{T}[\![\tau]\!]$; this interpretation is defined as expected by induction on the derivation tree (see [20] for details). For a well-typed closed formula φ of type Prop, a LTS $\mathscr{T} = (S, \delta)$ and a state $s \in S$, We write $s \models_{\mathscr{T}} \varphi$ if $s \in \mathscr{T}[\![\vdash \varphi : \mathsf{Prop}]\!]$.

Example 1 *Let* $\tau_3 = (\mathsf{Prop}^+ \to \mathsf{Prop})^+ \to \mathsf{Prop}^+ \to \mathsf{Prop}$ *and* $\varphi_3 =$

$$\left(\mu F^{\tau_3}.\, \lambda G^{\mathsf{Prop}^+ \to \mathsf{Prop}}, X^{\mathsf{Prop}}.\, (G\, X) \vee \left(F\ (\lambda Y^{\mathsf{Prop}}.G\, (G\, Y))\ X\right)\right) \quad (\lambda Z^{\mathsf{Prop}}.\, \langle a \rangle Z) \quad \langle b \rangle \top.$$

Then $s \models \varphi_3$ *iff there is* $n \geq 0$ *such that there is a path of the form* $a^{2^n} b$ *starting at* s. *Since* $\{a^{2^n} b \mid n \geq 0\}$ *is not a regular language, the property expressed by* φ_3 *cannot be expressed in the modal* μ*-calculus.*

Proposition 2 *[20] Let* $\mathscr{T} = (S, \delta)$ *be a LTS and let* $s, s' \in S$ *be two bisimilar states of* $\mathscr{T}$. *Then for any closed formula* φ *of type* Prop, $s \models_{\mathscr{T}} \varphi$ *iff* $s' \models_{\mathscr{T}} \varphi$.

We assume the standard notations $\wedge$, $[a]$ and $\nu X.\, (.)$ for the conjunction, the necessity modality, and the greatest fixed point, defined as the duals of $\vee$, $\langle a \rangle$ and $\mu X.\, (.)$ respectively.

Definition 1 (Negation Normal Form) *A HFL formula is in negation normal form if it is derivable from the grammar*

$$\varphi, \psi \ ::= \top \mid \bot \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid X \mid \lambda X^\sigma.\varphi \mid \varphi\ \psi \mid \mu X^\tau.\varphi \mid \nu X^\tau.\varphi$$

where the τ *are* monotone *types, i.e. types where all variances are equal to* $+$.

Note that since all variances are $+$, we omit them when writting formulas in negation normal form.

We say that two formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for every type environment Γ, for every LTS $\mathscr{T}$, for all type τ, the judgement $\Gamma \vdash \varphi : \tau$ is derivable iff $\Gamma \vdash \psi : \tau$ is, and in that case $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!] = \mathscr{T}[\![\Gamma \vdash \psi : \tau]\!]$.

Model-Checking We briefly recall the results known about the data complexity of HFL model-checking (see also the results of Lange *et al* on the combined complexity [1] or the descriptive complexity [14] of HFL and extensions).

Note that if $\mathscr{T} = (S, \delta)$ is a finite LTS, then for all type τ, the Boolean ring $\mathscr{T}[\![\tau]\!]$ is a finite set, and every element of $\mathscr{T}[\![\tau]\!]$ can be represented *in extension*. Moreover, the least fixed point of a monotone function $f : \mathscr{T}[\![\tau]\!] \to \mathscr{T}[\![\tau]\!]$ can be computed by iterating f at most n times, where n is the size of the finite boolean ring $\mathscr{T}[\![\tau]\!]$.

The order $\mathsf{ord}(\tau)$ of a type τ is defined as $\mathsf{ord}(\mathsf{Prop}) = 0$ and $\mathsf{ord}(\sigma^{\nu} \to \tau) = \max(\mathsf{ord}(\tau), 1 + \mathsf{ord}(\sigma))$. We write $\mathsf{HFL}(k)$ to denote the set of closed HFL formulas φ of type Prop such that all type annotations in φ are of order at most k. For every fixed $\varphi \in \mathsf{HFL}(k)$, we call $\mathsf{MC}(\varphi)$ the problem of deciding, given a LTS $\mathscr{T}$ and a state s of $\mathscr{T}$, wether $s \models_{\mathscr{T}} \varphi$.

Theorem 3 *[1] For every $k \geq 1$, for every $\varphi \in \mathsf{HFL}(k)$, the problem $\mathsf{MC}(\varphi)$ is in k-EXPTIME, and there is a $\psi_k \in \mathsf{HFL}(k)$ such that $\mathsf{MC}(\psi_k)$ is k-EXPTIME hard.*

3 Monotonization

In order to define a negation elimination procedure, the first idea is probably to reason like in the modal μ-calculus, and try to "push the negations to the leaves". Indeed, there are De Morgan laws for all logical connectives, including abstraction and application, since

$$\neg(\varphi\ \psi) \quad \equiv \quad (\neg\varphi)\ \psi \qquad\qquad \text{and} \qquad\qquad \neg(\lambda X^{\nu,\tau}.\psi) \quad \equiv \quad \lambda X^{-\nu,\tau}.\neg\psi.$$

In the modal μ-calculus, this idea is enough, because the "negation counting" criterion ensures that each pushed negation eventually reaches another negation and both anihilate. This does not happen for HFL. Consider for instance the formula $\varphi_4 =$

$$\left(\mu X^{\mathsf{Prop}^0 \to \mathsf{Prop}}.\ \lambda Y^{\mathsf{Prop},0}.\ (\neg Y) \vee \left(X\ (\langle a \rangle Y)\right)\right) \quad \top.$$

The negation already is at the leaf, but φ_4 is not in negation normal form. By fixed point unfolding, one can check that φ_4 is equivalent to the infinite disjunct $\bigvee_{n \geq 0} [a]^n \bot$, and thus could be expressed by $\mu X^{\mathsf{Prop}}.[a]X$. The generalization of this strategy for arbitrary formulas would be interesting, but it is unclear to us how it would be defined.

We follow another approach: we do not try to unfold fixed points nor to apply β-reductions during negation elimination, but we stick to the structure of the formula. In particular, in our approach a subformula denoting a function f is mapped to a subformula denoting a function f' in the negation normal form. Note that even if f is not monotone, f' must be monotone since it is a subformula of a formula in negation normal form. We call f' a *monotonization* of f.

Examples Before we formaly define monotonization, we illustrate its principles on some examples.

First, consider again the above formula φ_4. This formula contains the function $\lambda Y^{\mathsf{Prop},0}.\ (\neg Y) \vee \left(X\ (\langle a \rangle Y)\right)$. This function is unrestricted (neither monotone nor antitone). The monotonization of this

function will be the function $\lambda Y^{\mathsf{Prop},+},\overline{Y}^{\mathsf{Prop},+}.\overline{Y} \vee \big(X\ (\langle a\rangle Y)\big)$. To obtain this function, a duplicate $\overline{Y}$ of Y is introduced, and is used in place of $\neg Y$. Finally, the formula $\varphi_4' =$

$$\big(\mu X^{\mathsf{Prop}\to\mathsf{Prop}\to\mathsf{Prop}}.\ \lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ \overline{Y} \vee \big(X\ (\langle a\rangle Y)\ ([a]\overline{Y})\big)\big) \quad \top \qquad \bot$$

can be used as a negation normal form of φ_4. Note that the parameter $\top$ that was passed to the recursive function in φ_4 is duplicated in φ_4', with one duplicate that has been negated (the $\bot$ formula).

More generally, whenever a function is of type $\sigma^0 \to \tau$, we transform it into a function of type $\sigma_t^+ \to \sigma_t^+ \to \tau_t$ that takes two arguments of type σ_t (the translation of σ). Later, when this function is applied, we make sure that its argument is duplicated, one time positively, the other negatively.

Duplicating arguments might cause an exponential blow-up. For instance, for the formula $\varphi_5 =$

$$(\lambda X^{\mathsf{Prop}}.\ X \vee \langle a\rangle\neg X) \quad \big((\lambda Y^{\mathsf{Prop},0}.\ Y \vee \langle b\rangle\neg Y)\ \top\big)$$

if we duplicated arguments naively, we could get the formula $\varphi_5' =$

$$(\lambda X^{\mathsf{Prop}},\overline{X}^{\mathsf{Prop}}.\ X \vee \langle a\rangle\overline{X}) \quad \big((\lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ Y \vee \langle b\rangle\overline{Y})\ \top\ \bot\big) \quad \big((\lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ \overline{Y} \wedge [b]Y)\ \top\ \bot\big)$$

where the original $\top$ formula has been duplicated. If it occurred underneath $n+2$ applications of an unrestricted function, we would have 2^n copies of $\top$. We will come back to this problem in Section 4.

Let us now observe how monotonization works for functions that are antitone. In general, if f is an antitone function, both the "negation at the caller" $f_1(x) = \neg f(x)$ and the "negation at the callee" $f_2(x) = f(\neg x)$ are two monotone functions that faithfully represent f. Actually, both of them might be needed by our negation elimination procedure.

Consider the formula $\varphi_6 =$

$$(\lambda F^{\mathsf{Prop}^-\to\mathsf{Prop},+}.\mu X^{\mathsf{Prop}}.F\ (\neg X)) \quad (\lambda Y^{\mathsf{Prop},-}.\neg\langle a\rangle Y).$$

In order to compute the negation normal form of φ_6, we may represent $\lambda Y^{\mathsf{Prop},-}.\neg\langle a\rangle Y$ by its "negation at the callee", yielding the formula $\varphi_6' =$

$$(\lambda F^{\mathsf{Prop}\to\mathsf{Prop}}.\mu X^{\mathsf{Prop}}.F\ X) \quad (\lambda\overline{Y}^{\mathsf{Prop}}.[a]\overline{Y}).$$

Conversely, consider the formula $\varphi_7 =$

$$(\lambda F^{\mathsf{Prop}^-\to\mathsf{Prop},-}.\ \mu X^{\mathsf{Prop}}.\ (\neg F)\ X) \quad (\lambda Y^{\mathsf{Prop},-}.\ \neg\langle a\rangle Y).$$

The only difference with φ_6 is that the negation is now in front of F instead of X. In that case, "negation at the callee" does not help eliminating negations. But "negation at the caller" does, and yields the negation normal form $\varphi_7' =$

$$(\lambda\overline{F}^{\mathsf{Prop}\to\mathsf{Prop}}.\ \mu X^{\mathsf{Prop}}.\ \overline{F}\ X) \quad (\lambda Y^{\mathsf{Prop}}.\ \langle a\rangle Y).$$

These examples suggest a negation elimination that proceeds along possibly different strategies in the case of an application $\varphi\ \psi$, depending on the semantics of φ and ψ. In the next section, we explain how the strategy is determined by the type of φ. For now, we focus on making more formal our notion of monotonization.

$$
\begin{array}{ll}
\exp(\mathsf{Prop}) & = \mathsf{Prop} \\
\exp(\tau^+ \to \sigma) & = \exp(\tau)^+ \to \exp(\sigma) \\
\exp(\tau^- \to \sigma) & = \exp(\tau)^+ \to \exp(\sigma) \\
\exp(\tau^0 \to \sigma) & = \exp(\tau)^+ \to \exp(\tau)^+ \to \exp(\sigma)
\end{array}
\qquad
\begin{array}{ll}
\exp(\Gamma_1, \Gamma_2) & = \exp(\Gamma_1), \exp(\Gamma_2) \\
\exp(X^+ : \tau) & = X^+ : \exp(\tau) \\
\exp(X^- : \tau) & = \overline{X}^+ : \exp(\tau) \\
\exp(X^0 : \tau) & = X^+ : \exp(\tau), \overline{X}^+ : \exp(\tau)
\end{array}
$$

Figure 2: Expansion of types and typing environments towards monotonization.

Monotonization Relations We saw that our negation elimination bases on the ability to faithfully represent a predicate transformer φ by a monotone predicate transformer ψ; in this case, we will say that ψ is a *monotonization* of φ. We now aim at defining formally this notion. More precisely, we aim at defining the relation $\triangleleft$ such that $\varphi \triangleleft \psi$ holds if ψ is a monotonization of φ.

First of all, $\triangleleft$ relates a formula of type τ to a formula of type $\exp(\tau)$ as defined in Fig. 2: the number of arguments of φ is duplicated if φ is unrestricted, otherwise it remains the same, and of course ψ is monotone in all of its arguments.

In Fig. 2, we also associate to every typing environment Γ the typing environment $\exp(\Gamma)$ with all variances set to $+$, obtained after renaming all variables with variance $-$ in their bared version, and duplicating all variables with variance 0. In the remainder, we always implicitly assume that we translate formulas and typing environments that do not initially contain bared variables.

The relation $\triangleleft$ is then defined coinductively, in a similar way as logical relations for the λ-calculus. Let R be a binary relation among typing judgements of the form $\Gamma \vdash \varphi : \tau$. The relation R is well-typed if $(\Gamma \vdash \varphi : \tau)\, R\, (\Gamma' \vdash \varphi' : \tau')$ implies $\Gamma' = \exp(\Gamma)$ and $\tau' = \exp(\tau)$. When R is well typed, we write $\varphi R_{\Gamma,\tau} \varphi'$ instead of $(\Gamma \vdash \varphi : \tau)\, R\, (\Gamma' \vdash \varphi' : \tau')$.

Definition 2 *A binary relation R among typing judgements is a* monotonization relation *if it is well-typed, and for all formulas φ, φ', for all Γ, τ such that $\varphi\, R_{\Gamma,\tau}\, \varphi'$,*

1. *if φ, φ' are closed and $\tau = \mathsf{Prop}$, then $\varphi \equiv \varphi'$;*
2. *if $\Gamma = \Gamma', X^+ : \sigma$, then $(\lambda X^{\sigma,+}.\ \varphi)\, R_{\Gamma', \sigma^+ \to \tau}\, (\lambda X^{\exp(\sigma),+}.\ \varphi')$;*
3. *if $\Gamma = \Gamma', X^- : \sigma$, then $(\lambda X^{\sigma,-}.\ \varphi)\, R_{\Gamma', \sigma^- \to \tau}\, (\lambda \overline{X}^{\exp(\sigma),+}.\ \varphi')$;*
4. *if $\Gamma = \Gamma', X^0 : \sigma$, then $(\lambda X^{\sigma,0}.\ \varphi)\, R_{\Gamma', \sigma^0 \to \tau}\, (\lambda X^{\exp(\sigma),+}, \overline{X}^{\exp(\sigma),+}.\ \varphi')$;*
5. *if $\tau = \sigma^+ \to \upsilon$, then for all ψ, ψ' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi')$;*
6. *if $\tau = \sigma^- \to \upsilon$, then for all ψ, ψ', ψ'' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$ and $\psi' \equiv \neg\psi''$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi'')$;*
7. *if $\tau = \sigma^0 \to \upsilon$, then for all ψ, ψ', ψ'' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$ and $\psi' \equiv \neg\psi''$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi'\ \psi'')$.*

If $(R_i)_{i \in I}$ is a family of monotonization relation, then so is $\bigcup_{i \in I} R_i$; we write $\triangleleft$ for the largest monotonization relation.

Example 2 *Consider $\varphi = (\lambda X^{\mathsf{Prop},-}.\ \neg X)$. Then $\varphi \triangleleft_{\mathsf{Prop}^- \to \mathsf{Prop}} (\lambda \overline{X}^{\mathsf{Prop},+}.\ \overline{X})$. Consider also $\psi = (\lambda X^{\mathsf{Prop},0}.\ X \wedge \neg X)$. Then $\psi \triangleleft (\lambda X^{\mathsf{Prop},+}, \overline{X}^{\mathsf{Prop},+}.\ \bot)$ and $\psi \triangleleft (\lambda X^{\mathsf{Prop},+}, \overline{X}^{\mathsf{Prop},+}.\ X \wedge \overline{X})$.*

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{tr}_+(\top) & = & \top & \mathsf{tr}_+(\psi_1 \vee \psi_2) & = & \mathsf{tr}_+(\psi_1) \vee \mathsf{tr}_+(\psi_2) \\
\mathsf{tr}_-(\top) & = & \bot & \mathsf{tr}_-(\psi_1 \vee \psi_2) & = & \mathsf{tr}_-(\psi_1) \wedge \mathsf{tr}_-(\psi_2) \\
\mathsf{tr}_+(X) & = & X & \mathsf{tr}_v(\lambda X^{\tau,+}.\ \psi) & = & \lambda X^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_-(X) & = & \overline{X} & \mathsf{tr}_v(\lambda X^{\tau,-}.\ \psi) & = & \lambda \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_v(\neg\psi) & = & \mathsf{tr}_{-v}(\psi) & \mathsf{tr}_v(\lambda X^{\tau,0}.\ \psi) & = & \lambda X^{\mathsf{exp}(\tau)}, \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_+(\langle a\rangle\psi) & = & \langle a\rangle \mathsf{tr}_+(\psi) & \mathsf{tr}_+(\mu X^{\tau}.\ \psi) & = & \mu X^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_+(\psi) \\
\mathsf{tr}_-(\langle a\rangle\psi) & = & [a]\mathsf{tr}_-(\psi) & \mathsf{tr}_-(\mu X^{\tau}.\ \psi) & = & \nu \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_-(\psi)
\end{array}
$$

$$
\mathsf{tr}_v(\psi_1\ \psi_2) \ = \ \begin{cases} \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_+(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^+ \to \eta \\ \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_-(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^- \to \eta \\ \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_+(\psi_2)\ \mathsf{tr}_-(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^0 \to \eta \end{cases}
$$

Figure 3: Type-Directed Negation Elimination

4 Negation Elimination

Our negation elimination procedure proceeds in two steps: first, a formula φ is translated into a formula $\mathsf{tr}_+(\varphi)$ that denotes the monotonization of φ; then, $\mathsf{tr}_+(\varphi)$ is concisely represented in order to avoid an exponential blow-up.

The transformation $\mathsf{tr}_+(.)$ is presented in Figure 3. The transformation proceeds by structural induction on the formula, and is defined as a mutual induction with the companion transformation $\mathsf{tr}_-(.)$. Whenever a negation is encountered, it is eliminated and the dual transformation is used. As a consequence, wether $\mathsf{tr}_+(.)$ or $\mathsf{tr}_-(.)$ should be used for a given subformula depends on the polarity [15] of this subformula.

Lemma 4 *Let φ be a fixed closed formula of type* Prop. *For every subformula ψ of φ, let $\mathsf{tr}_+(\psi)$ and $\mathsf{tr}_-(\psi)$ be defined as in Figure 3, and let $\Gamma \vdash \psi : \tau$ be the type judgement associated to ψ in the type derivation of φ. Then the following statements hold.*

1. $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi) : \mathsf{exp}(\tau)$ *and* $\mathsf{exp}(\neg\Gamma) \vdash \mathsf{tr}_-(\psi) : \mathsf{exp}(\tau)$.
2. $\psi \lhd_{\Gamma,\tau} \mathsf{tr}_+(\psi)$ *and* $\psi \lhd_{\Gamma,\tau} \neg\mathsf{tr}_-(\psi)$.

Proof: By induction on ψ. We only detail the point 1 in the case of $\psi = \psi_1\ \psi_2$ with $\mathsf{type}(\psi_1/\varphi) = \sigma^- \to \tau$. Let us assume the two statements hold for ψ_1 and ψ_2 by induction hypothesis. Let Γ be such that $\Gamma \vdash \psi : \tau$, $\Gamma \vdash \psi_1 : \sigma^- \to \tau$, and $\neg\Gamma \vdash \psi_2 : \sigma$. By induction hypothesis, the judgements $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi_1) : \mathsf{exp}(\sigma^- \to \tau)$ and $\mathsf{exp}(\neg\neg\Gamma) \vdash \mathsf{tr}_-(\psi_2) : \mathsf{exp}(\sigma)$ are derivable. Since $\mathsf{exp}(\sigma^- \to \tau) = \mathsf{exp}(\sigma)^+ \to \mathsf{exp}(\tau)$ and $\neg\neg\Gamma = \Gamma$, the typing rule for function application in the monotone case of Fig. 1 yields $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi_1)\ \mathsf{tr}_-(\psi_2) : \mathsf{exp}(\tau)$, which shows statement 1 for $\mathsf{tr}_+(.)$. The case for $\mathsf{tr}_-(.)$ is similar. □

Corollary 5 *If φ is a closed formula of type* Prop, *then $\varphi \equiv \mathsf{tr}_+(\varphi)$ and $\mathsf{tr}_+(\varphi)$ is in negation normal form.*

As observed in Section 3, the duplication of the arguments in the case $v = 0$ of the monotonization of $\varphi\psi$ may cause an exponential blow-up in the size of the formula. However, this blow-up does not happen if we allow some sharing of identical subformulas.

Let φ be a fixed closed formula. We say that two subformulas ψ_1 and ψ_2 of φ are identical if they are syntactically equivalent and if moreover they have the same type and are in a same typing context,

i.e. if the type derivation of φ goes through the judgements $\Gamma_i \vdash \psi_i : \tau_i$ for syntactically equivalent Γ_i and τ_i. For instance, in the formula

$$(\lambda X^{\mathsf{Prop}\to\mathsf{Prop}}.\, X) \quad \big((\lambda X^{(\mathsf{Prop}\to\mathsf{Prop})\to(\mathsf{Prop}\to\mathsf{Prop})}.\, X) \quad \big((\lambda Y^{\mathsf{Prop}\to\mathsf{Prop}}.Y)\ \top\big)\big)$$

any two distinct subformulas are not identical (including the subformulas restricted to X). We call *dag size* of φ the number of non-identical subformulas of φ.

Lemma 6 *There is a logspace computable function* $\mathsf{share}(.)$ *that associates to every closed formula* φ *of dag size n a closed formula* $\mathsf{share}(\varphi)$ *of tree size* $\mathscr{O}(n\cdot|\mathsf{vars}(\varphi)|)$ *such that* $\varphi \equiv \mathsf{share}(\varphi)$.

Proof: Let φ be fixed, and let $\varphi_1\ldots,\varphi_n$ be an enumeration of all subformulas of φ such that if φ_i is a strict subformula of φ_j, then $i<j$. In particular, we must have $\varphi=\varphi_n$. Pick some fresh variables $X_1,X_2,\ldots,X_n \in \mathsf{Var}$ and let $\upsilon_i = \mathsf{type}(\varphi_i/\varphi)$. For every $i=1,\ldots n$, let $Y_1,\sigma_1,\nu_1,\ldots Y_k,\sigma_k,\nu_k$ be a fixed enumeration of the free variables of φ_i, their types and their variances, and let $\lambda_i(\psi) = \lambda Y_1^{\sigma_1,\nu_1},\ldots,Y_k^{\sigma_k,\nu_k}.\ \psi$ and $@_i(\psi) = \psi\ Y_1\ \ldots\ Y_k$. Finally, let $\tau_i = \sigma_1^{\nu_1}\to\ldots\sigma_k^{\nu_k}\to\upsilon_i$. For every subformula ψ of φ, let $\|\psi\|$ be defined by case analysis on the first logical connective of ψ:

- if $\psi=\varphi_i=\eta Y^\sigma.\ \varphi_j$, where $\eta\in\{\lambda,\mu,\nu\}$, then $\|\psi\| = \lambda_i\big(\eta Y^\sigma.\ @_j(X_j)\big)$;
- if $\psi=\varphi_i=\varphi_j\oplus\varphi_k$, where $\oplus\in\{\vee,\wedge,\text{application}\}$, then $\|\psi\| = \lambda_i\big(@_j(X_j)\oplus @_k(X_k)\big)$;
- if $\psi=\varphi_i=\spadesuit\varphi_j$, where $\spadesuit\in\{\neg,\langle a\rangle,[a]\}$, then $\|\psi\| = \lambda_i\big(\spadesuit(@_j(X_j))\big)$;
- otherwise $\|\varphi_i\| = \lambda_i(\varphi_i)$.

Finally, let $\mathsf{share}(\varphi) = \textbf{let}\ X_1^{\tau_1} = \|\varphi_1\|\ \textbf{in let}\ X_2^{\tau_2} = \|\varphi_2\|\ \textbf{in}\ \ldots\ \textbf{let}\ X_{n-1}^{\tau_{n-1}} = \|\varphi_{n-1}\|\ \textbf{in}\ \|\varphi_n\|$ where $\textbf{let}\ X^\tau=\psi\ \textbf{in}\ \psi'$ is a macro for $(\lambda X^\tau.\ \psi')\ \psi$. Then $\mathsf{share}(\varphi)$ has the desired properties. □

Theorem 7 *There is a logspace-computable function* $\mathsf{nnf}(.)$ *that associates to every closed HFL formula* φ *(without variable masking) of type* Prop *a closed formula* $\mathsf{nnf}(\varphi)$ *such that*

1. $\varphi\equiv\mathsf{nnf}(\varphi)$,
2. $\mathsf{nnf}(\varphi)$ *is in negation normal form, and*
3. $|\mathsf{nnf}(\varphi)| = \mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$,

where $|\psi|$ *denotes the size of the tree representation of* ψ *(i.e. the number of symbols in* ψ*), and* $\mathsf{vars}(\varphi) = \mathsf{fv}(\varphi)\cup\mathsf{bv}(\varphi)$ *is the set of variables that occur in* φ.

Proof: Let $\mathsf{nnf}(\varphi)=\mathsf{share}(\mathsf{tr}_+(\varphi))$. This function is logspace computable ($\mathsf{tr}_+(\varphi)$ can be computed "on-the-fly") and $\mathsf{nnf}(\varphi)$ is of size $\mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$ by Figure 3 and Lemma 6. The formula $\mathsf{tr}_+(\varphi)$ is in negation normal form, and $\mathsf{share}(.)$ does not introduce new negations, so $\mathsf{nnf}(\varphi)$ is in negation normal form. Looking back at Figure 3, it can be checked that its dag size is linear in the dag size of φ, so the tree size of $\mathsf{nnf}(\varphi)$ is linear in the tree size of φ. Moreover, $\mathsf{nnf}(\varphi)\equiv\mathsf{tr}_+(\varphi)$ by Lemma 6, and $\mathsf{tr}_+(\varphi)\equiv\varphi$ by Corollary 5. □

5 Conclusion

We have considered the higher-order modal fixed point logic [20] (HFL) and its fragment without negations, and we have shown that both formalisms are equally expressive. More precisely, we have defined a procedure for transforming any closed HFL formula φ denoting a state predicate into an equivalent formula $\mathsf{nnf}(\varphi)$ without negations of size $\mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$. The procedure works in two phases: in a

first phase, a transformation we called *monotonization* eliminates all negations and represents arbitrary functions of type $\tau \to \sigma$ by functions of type $\tau \to \tau \to \sigma$ by distinguishing positive and negative usage of the function parameter. The price to pay for this transformation is an exponential blow-up in the size of the formula. If the formula is represented as a circuit, however, the blow-up is only linear. The second phase of our negation elimination procedure thus consists in implementing the sharing of common subformulas using higher-orderness. Thanks to this second phase, our procedure yields a negation-free formula $\mathsf{nnf}(\varphi)$ of size $\mathscr{O}(\mathsf{size}(\varphi) \cdot |\mathsf{vars}(\varphi)|)$, hence quadratic in the worst case in the size of the original formula φ.

Typed versus Untyped Negation Elimination Our monotonization procedure is *type-directed*: the monotonization of $\varphi\ \psi$ depends on the variance of φ, that is statically determined by looking at the type of φ. One might wonder if we could give a negation elimination that would not be type-directed. A way to approach this question is to consider an untyped conservative extension of the logic where we do not have to care about the existence of the fixed points – for instance, one might want to interprete $\mu X.\varphi(X)$ as the inflationary "fixed point" [7]. We believe that we could adapt our monotonization procedure to this setting, and it would indeed become a bit simpler: we could always monotonize $\varphi\ \psi$ "pessimistically", as if φ were neither a monotone nor an antitone function. For instance, the formula $\mu X.(\lambda Y.Y)\ X$ would be translated into $\mu X.(\lambda Y,\overline{Y}.Y)\ X\ \neg X$.

In our typed setting, it is crucial to use the type-directed monotonization we developed, because monotizing pessimistically might yield ill-typed formulas. In an untyped setting, a pessimistic monotonization is possible, but it yields less concise formulas, and it looses the desirable property that $\mathsf{nnf}(\mathsf{nnf}(\varphi)) = \mathsf{nnf}(\varphi)$.

So types, and more precisely variances, seem quite unavoidable. However, strictly speaking, the monotonization we introduced is *variance-directed*, and not really type-directed. In particular, our monotonization might be extended to the untyped setting, relying on some other static analysis than types to determine the variances of all functional subformulas.

Sharing and Quadratic Blow-Up The idea of sharing subterms of a λ-term is reminiscent to implementations of λ-terms based on hash-consing [8, 11] and to compilations of the λ calculus into interaction nets [13, 16, 10]. We showed how sharing can be represented directly in the λ-calculus, whereas hash-consing and interaction nets are concerned with representing sharing either in memory or as a circuit. We compile typed λ-terms into typed λ-terms; a consequence is that we do not manage to share subterms that are syntactically identical but have either different types or are typed using different type assumptions for their free variables. This is another difference with hash consing and interaction nets, where syntactic equality is enough to allow sharing subterms. It might be the case that we could allow more sharing if we did not compile into a simply typed λ-calculus but in a ML-like language with polymorphic types.

An interesting issue is the quadratic blow-up of our implementation of "λ-circuits". One might wonder wether a more succinct negation elimination is possible, in particular a negation elimination with linear blow-up. To answer this problem, it would help to answer the following simpler problem: *given a λ-term t with n syntactically distinct subterms, is there an effectively computable λ-term t' of size $\mathscr{O}(n)$ such that $t =_{\beta\eta} t'$*? We leave that problem for future work.

References

[1] Roland Axelsson, Martin Lange & Rafal Somla (2007): *The Complexity of Model Checking Higher-Order Fixpoint Logic*. *Logical Methods in Computer Science* 3(2), doi:10.2168/LMCS-3(2:7)2007.

[2] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague & Olivier Serre (2013): *C-SHORe: a collapsible approach to higher-order verification*. In: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pp. 13–24, doi:10.1145/2500365.2500589.

[3] Florian Bruse (2014): *Alternating Parity Krivine Automata*. In Erzsbet Csuhaj-Varj, Martin Dietzfelbinger & Zoltn sik, editors: *Mathematical Foundations of Computer Science 2014*, *Lecture Notes in Computer Science* 8634, Springer Berlin Heidelberg, pp. 111–122, doi:10.1007/978-3-662-44522-8_10.

[4] Thierry Cachat (2003): *Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games*. In: *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, pp. 556–569, doi:10.1007/3-540-45061-0_45.

[5] Arnaud Carayol & Olivier Serre (2012): *Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection*. In: *LICS*, pp. 165–174, doi:10.1109/LICS.2012.73.

[6] Werner Damm (1982): *The IO- and OI-hierarchies*. *Theoretical Computer Science* 20(2), pp. 95 – 207, doi:10.1016/0304-3975(82)90009-3.

[7] Anuj Dawar, Erich Grädel & Stephan Kreutzer (2004): *Inflationary fixed points in modal logic*. *ACM Trans. Comput. Log.* 5(2), pp. 282–315, doi:10.1145/976706.976710.

[8] Jean-Christophe Filliâtre & Sylvain Conchon (2006): *Type-safe Modular Hash-consing*. In: *Proceedings of the 2006 Workshop on ML*, ML '06, ACM, New York, NY, USA, pp. 12–19, doi:10.1145/1159876.1159880.

[9] Koichi Fujima, Sohei Ito & Naoki Kobayashi (2013): *Practical Alternating Parity Tree Automata Model Checking of Higher-Order Recursion Schemes*. In: *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, pp. 17–32, doi:10.1007/978-3-319-03542-0_2.

[10] Georges Gonthier, Martín Abadi & Jean-Jacques Lévy (1992): *The Geometry of Optimal Lambda Reduction*. In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, ACM, New York, NY, USA, pp. 15–26, doi:10.1145/143165.143172.

[11] Jean Goubault (1993): *Implementing Functional Languages with Fast Equality Sets and Maps: an Exercise in Hash Cons*. In: *Journées Francophones des Langages Applicatifs (JFLA'93)*, Annecy, pp. 222–238.

[12] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong & Olivier Serre (2008): *Collapsible Pushdown Automata and Recursion Schemes*. In: *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pp. 452–461, doi:10.1109/LICS.2008.34.

[13] John Lamping (1990): *An Algorithm for Optimal Lambda Calculus Reduction*. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, ACM, New York, NY, USA, pp. 16–30, doi:10.1145/96709.96711.

[14] Martin Lange & Étienne Lozes (2014): *Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic*. In: *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, pp. 90–103, doi:10.1007/978-3-662-44602-7_8.

[15] Olivier Laurent (2002): *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II.

[16] Ian Mackie (1998): *YALE: Yet Another Lambda Evaluator Based on Interaction Nets*. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ICFP '98, ACM, New York, NY, USA, pp. 117–128, doi:10.1145/289423.289434.

[17] A. N. Maslov. (1976): *Multilevel stack automata*. *Problems of Information Transmission* 12, pp. 38–43.

[18] Sylvain Salvati & Igor Walukiewicz (2014): *Krivine machines and higher-order schemes*. *Inf. Comput.* 239, pp. 340–355, doi:10.1016/j.ic.2014.07.012.

[19] Taku Terao & Naoki Kobayashi (2014): *A ZDD-Based Efficient Higher-Order Model Checking Algorithm*. In: *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, pp. 354–371, doi:10.1007/978-3-319-12736-1_19.

[20] Mahesh Viswanathan & Ramesh Viswanathan (2004): *A Higher Order Modal Fixed Point Logic*. In Ph. Gardner & N. Yoshida, editors: *CONCUR, Lecture Notes in Computer Science* 3170, Springer, pp. 512–528, doi:10.1007/978-3-540-28644-8_33.

Reasoning about modular datatypes with Mendler induction

Paolo Torrini Tom Schrijvers

Department of Computer Science, KU Leuven, Belgium
{p.torrini, tom.schrijvers}@cs.kuleuven.be

In functional programming, datatypes à la carte provide a convenient modular representation of recursive datatypes, based on their initial algebra semantics. Unfortunately it is highly challenging to implement this technique in proof assistants that are based on type theory, like Coq. The reason is that it involves type definitions, such as those of type-level fixpoint operators, that are not strictly positive. The known work-around of impredicative encodings is problematic, insofar as it impedes conventional inductive reasoning. Weak induction principles can be used instead, but they considerably complicate proofs.

This paper proposes a novel and simpler technique to reason inductively about impredicative encodings, based on Mendler-style induction. This technique involves dispensing with dependent induction, ensuring that datatypes can be lifted to predicates and relying on relational formulations. A case study on proving subject reduction for structural operational semantics illustrates that the approach enables modular proofs, and that these proofs are essentially similar to conventional ones.

1 Introduction

Developing high-quality software artifacts, including programs as well as programming languages, can be very expensive, and so can formally proving their properties. This makes it highly desirable to maximise reuse and extensibility. Modularity plays an essential role in this context: a component is modular whenever it can be specified independently of the whole collection – therefore, a modular characterisation of an artifact implies that its extension does not require changes to what is already in stock.

In functional programming, it is natural to rely on a structured characterisation of components based on recursive datatypes. However, conventional datatypes are not extensible – each one fixes a closed set of constructors with respect to which case analysis may have to be exhaustive, hence each case implicitly depends on the whole collection. An elegant solution to this tension between structural characterisation and modularity, also known as the *expression problem*, has been found with the notion of *modular datatype* (MDT) – i.e., datatypes *à la carte*, introduced in Haskell by Swierstra [16]. The definition of an MDT consists of two distinct parts: the grammar, as a non-recursive structure based on a functor, and the recursive datatype, as the recursive closure of the functor by a type-level fixed point. Grammar functors behave as modules, as they can be defined independently and combined together by coproduct.

In Haskell, an MDT can be easily implemented in terms of conventional datatypes, which can be used to define the grammar as well as the recursive closure (as recalled in Section 2). However, Haskell's datatype definition of the type-level fixpoint operator is not strictly positive, and therefore it is problematic from the point of view of less liberal type systems. As a general-purpose programming language, Haskell relies on types that do not enforce totality (i.e., either termination or productivity). This makes type checking easier in the presence of non-termination. Unfortunately, allowing for non-total programs can lead to inconsistency under a program-as-proof interpretation. For this reason, proof assistants based on the Curry-Howard correspondence are usually based on more restrictive type systems. Proof assistants such as Coq, Agda, Isabelle and Twelf, for instance, rely on a syntactic criterion of monotonicity

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 143–157, doi:10.4204/EPTCS.191.13

which ensures totality, by requiring that all the occurrences of an inductive datatype in its definition are strictly positive – hence incompatibly with the Haskell-style representation of MDTs.

Coq is a theorem prover based on the calculus of inductive constructions (CIC) [3] which extends the calculus of constructions (CC) [5] with inductive and coinductive definitions. CC, the most expressive system of the lambda cube [2], allows for types depending on terms, type-level functions and full parametric polymorphism, hence also for definitions that are impredicative, in the sense of referring in their bodies to collections that are being defined. One of the main approaches to represent MDT in Coq, due to Delaware, Oliveira and Schrijvers [7] and implemented in the MTC/3MT framework [6], takes advantage of impredicativity, and relies on the Church encoding of fixed points (as recalled in Section 3). Another promising approach, due to Keuchel and Schrijvers [11], relies on containers – it is predicative, but it involves a more indirect representation of types. Church encodings are purely based on CC and do not involve any extra-logical machinery – however, they rather complicate inductive reasoning. Impredicative definitions have an eliminative character that hides term structure, hence making it harder to reason by induction. The solution proposed by Delaware *et al.* is quite general – however, it relies on proof algebras that pack terms together with proofs using Σ-types, and this leads to inductive proofs that have a significant overhead with respect to the conventional, non-modular ones.

This paper proposes a novel solution to the problem of reasoning inductively with impredicatively encoded MDT, based on the use of Mendler-style induction [12, 18, 1]. Mendler's characterisation of iteration makes it possible to encode an induction principle within the impredicative encoding of an MDT. Unlike Delaware *et al.*, we use Mendler algebras as proof algebras. This leads to inductive proofs that are straightforwardly modular and ultimately closer to conventional ones (Section 4). Although this approach cannot handle dependent induction, this limitation is of little consequence as long as we are reasoning about relational formulations. Nonetheless, this may make it necessary to lift inductive datatypes to inductively defined predicates, in order to use them as inductive arguments in proofs.

In order to reason inductively on relations, we clearly need to rely on functor shapes that can represent them as well as mutual dependencies. Such need is highlighted throughout a case study on the formalisation of a language based on structural operational semantics (Section 5, Coq implementation available [17]). The language, for which we prove type preservation, has a definition that involves mutual dependency between expressions and declarations.

2 Datatypes a-la-carte

MDTs as introduced by Swierstra [16] are essentially a functional programming application of the initial algebra semantics of inductive types. This consists of associating an inductive datatype to an endofunctor in a base category, then interpreting it as the initial object in the category of algebras determined by the functor [9, 19].

In its simplest form, taking sets (S) as the base category, each inductive datatype $\rho : \mathsf{S}$ can be associated with a covariant endofunctor (*signature functor*), i.e. a map $F : \mathsf{S} \to \mathsf{S}$ for which there exists a map (*functor map*) $\mathsf{fmap}_F\ \{A\ B\} :\ (A \to B) \to (F\ A \to F\ B)$ that preserves identities and composition, with $A, B : \mathsf{S}$ (always treated as implicit parameters). Semantically, an algebra determined by F (F*-algebra*) is a pair $\langle C, \phi \rangle$ where $C : \mathsf{S}$ is the *carrier* and $\phi : F\ C \to C$ is the *structure map*. $F\ C$ can be understood as the denotation of a grammar based on signature F, given carrier C. The initial object $\langle \mu F, \mathsf{in}_F \rangle$, where in_F is an isomorphism and thus has an inverse out_F, gives the denotation of ρ obtained as the fixpoint closure of F. In this way, the non-recursive structural characterisation of ρ, which essentially corresponds to case analysis, is separated from its recursive closure. For instance, in a functional language which allows

for datatype definitions with data constructors and Haskell-style destructors (while we mainly rely on Coq-style and standard algebraic notation), the following

$$\mathsf{dt\_def}\ \rho\ =\ \mathsf{c}_1\ (\tau_1[\rho/A])\mid\ldots\mid\mathsf{c}_k\ (\tau_k[\rho/A]) \tag{1}$$

can be decomposed in

$$\mathsf{dt\_def}\ F\ A\ =\ \mathsf{c}_1\ (\tau_1)\mid\ldots\mid\mathsf{c}_k\ (\tau_k) \tag{2}$$

and

$$\rho\ =_{df}\ \mathsf{Fix}\ F \tag{3}$$

where $\mathsf{Fix}\ F$ is the syntactic representation of μF, i.e.

$$\mathsf{dt\_def}\ \mathsf{Fix}\ F\ =\ \mathsf{in}\ (\mathsf{out}:\ F\ (\mathsf{Fix}\ F)) \tag{4}$$

For each F-algebra $\langle C,f\rangle$, the unique incoming algebra morphism from the initial algebra is determined by the unique *mediating map* $\mathsf{fold}_{F,C,f}:\mu F\to C$. Syntactically, this corresponds to the definition of $\mathsf{fold}\ F\ C:\ (F\ C\to C)\to(\mathsf{Fix}\ F\to C)$ as a recursive function.

$$\mathsf{fold}\ F\ C\ f\ x\ =_{df}\ f\ (\mathsf{fmap}\ F\ (\mathsf{fold}\ F\ C\ f)\ (\mathsf{out}\ x)) \tag{5}$$

Functors are composable by coproduct (+), i.e., if $F_1,F_2:\mathsf{S}\to\mathsf{S}$ are functors, so is $F_1{+}F_2$, with

$$\mathsf{dt\_def}\ (F_1{+}F_2)\ C\ =\ \mathsf{inl}\ (F_1\ C)\mid\mathsf{inr}\ (F_2\ C) \tag{6}$$

This results in a modular definition of the inductive datatype $\mathsf{Fix}\ (F_1{+}F_2)$ – not to be confused with $\mathsf{Fix}\ F_1{+}\mathsf{Fix}\ F_2$. In connection with coproducts, Haskell implementations of MDTs rely on type classes to automate injections and projections, using smart constructors and class constraints to express subsumption between functors. As a concrete example, following Swierstra [16], the conventional datatype

$$\mathsf{dt\_def}\ \mathsf{Trm}\ =\ \mathsf{lit}\ (\mathsf{Int})\mid\mathsf{add}\ (\mathsf{Trm}*\mathsf{Trm}) \tag{7}$$

can be decomposed into two modules

$$\mathsf{dt\_def}\ \mathsf{Trm}_{\mathsf{G1}}\ C\ =\ \mathsf{lit}\ (\mathsf{Int})\qquad\qquad\mathsf{dt\_def}\ \mathsf{Trm}_{\mathsf{G2}}\ C\ =\ \mathsf{add}\ (C*C) \tag{8}$$

and thus modularly defined:

$$\mathsf{Trm}_{\mathsf{G}}\ =_{df}\ \mathsf{Trm}_{\mathsf{G1}}+\mathsf{Trm}_{\mathsf{G2}}\qquad\qquad\mathsf{Trm}\ =_{df}\ \mathsf{Fix}\ \mathsf{Trm}_{\mathsf{G}} \tag{9}$$

Moreover, given a notion of value and a conventional recursive definition of evaluation

$$\begin{array}{ll}\mathsf{dt\_def}\ \mathsf{Val}\ =\ \mathsf{val}\ (\mathsf{vv}:\mathsf{Int}) & \mathsf{eval}:\mathsf{Trm}\to\mathsf{Val}\\ & \mathsf{eval}\ (\mathsf{lit}\ x)\ =_{df}\ \mathsf{val}\ x\\ & \mathsf{eval}\ (\mathsf{add}\ (e_1,e_2))\ =_{df}\ \mathsf{val}\ ((\mathsf{vv}\circ\mathsf{eval}\ e_1)+(\mathsf{vv}\circ\mathsf{eval}\ e_2))\end{array} \tag{10}$$

the latter can be represented by an algebra and modularly decomposed as follows, allowing for a modular definition of the dynamic semantics.

$$\begin{array}{ll}\mathsf{eval}_{\mathsf{G1}}:\mathsf{Trm}_{\mathsf{G1}}\ \mathsf{Val}\to\mathsf{Val} & \mathsf{eval}_{\mathsf{G1}}\ (\mathsf{lit}\ x)\ =_{df}\ \mathsf{val}\ x\\ \mathsf{eval}_{\mathsf{G2}}:\mathsf{Trm}_{\mathsf{G2}}\ \mathsf{Val}\to\mathsf{Val} & \mathsf{eval}_{\mathsf{G2}}\ (\mathsf{add}\ (x_1,x_2))\ =_{df}\ \mathsf{val}\ ((\mathsf{vv}\ x_1)+(\mathsf{vv}\ x_2))\\ \mathsf{eval}_{\mathsf{G}}:\mathsf{Trm}_{\mathsf{G}}\ \mathsf{Val}\to\mathsf{Val} & \mathsf{eval}_{\mathsf{G}}\ (\mathsf{inl}\ e)\ =_{df}\ \mathsf{eval}_{\mathsf{G1}}\ e\\ & \mathsf{eval}_{\mathsf{G}}\ (\mathsf{inr}\ e)\ =_{df}\ \mathsf{eval}_{\mathsf{G2}}\ e\end{array} \tag{11}$$

$$\mathsf{eval}\ e\ =_{df}\ \mathsf{fold}\ \mathsf{Trm}_{\mathsf{G}}\ \mathsf{Val}\ \mathsf{eval}_{\mathsf{G}}\ e \tag{12}$$

3 Impredicative encoding

The MDT representation discussed so far works well with Haskell, but not with Coq. Representing F as an inductive datatype is not problematic, but this is not so for the fixpoint closure. Since the constructor of $\mathsf{Fix}\,F$ has type $F\ (\mathsf{Fix}\ \mathsf{F}) \to \mathsf{Fix}\,F$, the datatype has a non-strictly positive occurrence in its definition, as parameter of the argument type – hence it is rejected by Coq. There is an analogous issue with the definition of fold, which is not structurally recursive. The solution to this problem adopted by Delaware *et al.* in [7], which we summarise here, goes back to Pfenning and Paulin-Mohring [13] in relying on a Church-style encoding of fixpoint operators, thus requiring impredicative definitions.

From the point of view of a type theoretic representation, the type of an algebra (that we may call *Church algebra*, or conventional algebra) can be identified with the type of its structure map.

$$\mathsf{Alg}^{\mathsf{C}}\ F\ C\ =_{df}\ F\ C \to C \tag{13}$$

If the initiality property of fixed points is weakened to an existence property, a fixpoint operator can be regarded as a function that maps an algebra to its carrier. An abstract definition of the type-level fixpoint operator $\mathsf{Fix}^{\mathsf{C}} : (\mathsf{S} \to \mathsf{S}) \to \mathsf{S}$ can then be given, as elimination rule for F-algebras, impredicatively with respect to S (this requires the impredicative set option in Coq, as used in MTC/3MT [7]).

$$\mathsf{Fix}^{\mathsf{C}}\ F =_{df} \forall A : \mathsf{S}.\ \mathsf{Alg}^{\mathsf{C}}\ F\ A \to A \tag{14}$$

The map $\mathsf{fold}^{\mathsf{C}}\ F\ C : \mathsf{Alg}^{\mathsf{C}}\ F\ C \to \mathsf{Fix}^{\mathsf{C}}\ F \to C$, corresponding to the elimination of a fixpoint value, can now be defined as the application of that value.

$$\mathsf{fold}^{\mathsf{C}}\ F\ C\ f\ x =_{df} x\ C\ f \tag{15}$$

Relying on the functoriality of F, the in-map $\mathsf{in}^{\mathsf{C}}\ F : F(\mathsf{Fix}\,F) \to \mathsf{Fix}\,F$ and the out-map $\mathsf{out}^{\mathsf{C}}\ F : \mathsf{Fix}\,F \to F(\mathsf{Fix}\,F)$ can be defined as functions.

$$\mathsf{in}^{\mathsf{C}}\ F\ =_{df}\ \lambda x\ A\ f.\ f(\mathsf{fmap}\ F\ (\mathsf{fold}^{\mathsf{C}}\ F\ A\ f)\ x) \tag{16}$$

$$\mathsf{out}^{\mathsf{C}}\ F\ =_{df}\ \mathsf{fold}^{\mathsf{C}}\ F\ (F(\mathsf{Fix}\,F))\ (\mathsf{fmap}\ F\ (\mathsf{in}^{\mathsf{C}}\ F)) \tag{17}$$

Notice that the definition of $\mathsf{fold}^{\mathsf{C}}\ F\ C\ f$ does not guarantee the uniqueness of the mediating map – it rather corresponds to a condition called quasi-initiality by Wadler [19]. In order to obtain uniqueness, hence to ensure that in^{C} is an isomorphism, the following implication needs to be proved for F [7, 11, 10].

$$(\forall x : \mathsf{Fix}^{\mathsf{C}}\ F.\ h\ (\mathsf{in}^{\mathsf{C}}\ F\ x)\ =\ f\ (\mathsf{fmap}\ F\ h\ x))\ \to\ (h\ =\ \mathsf{fold}^{\mathsf{C}}\ F\ C\ f) \tag{18}$$

Semantically, the impredicative encoding of the fixed points is closely associated with a constructor, usually called build, that allows for an alternative interpretation of inductive datatypes in terms of limit constructions, provably equivalent to the initial algebra semantics [8].

3.1 Indexed algebras

A relation can be represented as a function from the type of its tupled arguments to the type P of propositions. From the point of view of initial semantics, assuming P can be represented as a category, the modular representation of inductively defined relations only requires a shift of base category. Given a type K (i.e., K : Type) and assuming it can be represented as a small category, we can take the category

of diagrams of type K in P as the base category for the relations of type $K \to \mathsf{P}$. In such category, an endofunctor $R : (K \to \mathsf{P}) \to (K \to \mathsf{P})$ that here we call *indexed functor*, is then associated with a map (*indexed functor map*) that preserves identities and composition.

$$\mathsf{fmap}^{\mathsf{I}}\ K\ R : \forall\ \{A\ B :\ K \to \mathsf{P}\}.\ (\forall w : K.\ A\ w \to B\ w) \to (\forall w : K.\ R\ A\ w \to R\ B\ w) \tag{19}$$

From the point of view of the impredicative encoding, an R-algebra can be characterised as an indexed map, given a carrier $D : K \to \mathsf{P}$.

$$\mathsf{Alg}^{\mathsf{CI}}\ K\ R\ D\ =_{df}\ \forall w : K.\ R\ D\ w \to D\ w \tag{20}$$

The corresponding fixpoint operator has type $((K \to \mathsf{P}) \to K \to \mathsf{P}) \to K \to \mathsf{P}$.

$$\mathsf{Fix}^{\mathsf{CI}}\ K\ R\ (w : K)\ =_{df}\ \forall A : K \to \mathsf{P}.\ \mathsf{Alg}^{\mathsf{CI}}\ K\ R\ A\ \to\ A\ w \tag{21}$$

The structuring operators can be defined as follows:

$$\mathsf{fold}^{\mathsf{CI}}\ K\ R :\ \forall A\ (f : \mathsf{Alg}^{\mathsf{CI}}\ K\ R\ A)\ (w : K).\ \mathsf{Fix}^{\mathsf{CI}}\ K\ R\ w \to A\ w\ =_{df}\ \lambda A\ f\ w\ e.\ e\ A\ f \tag{22}$$

$$\begin{array}{l}\mathsf{in}^{\mathsf{CI}}\ K\ R\ (w : K) : R\ (\mathsf{Fix}^{\mathsf{CI}}\ K\ R)\ w \to \mathsf{Fix}^{\mathsf{CI}}\ K\ R\ w\ =_{df} \\ \qquad \lambda x\ A\ f.\ f\ w\ (\mathsf{fmap}^{\mathsf{I}}\ K\ R\ (\mathsf{fold}^{\mathsf{CI}}\ K\ R\ A\ f)\ w\ x)\end{array} \tag{23}$$

$$\begin{array}{l}\mathsf{out}^{\mathsf{CI}}\ K\ R\ (w : K) : \mathsf{Fix}^{\mathsf{CI}}\ K\ R\ w \to R\ (\mathsf{Fix}^{\mathsf{CI}}\ K\ R)\ w\ =_{df} \\ \qquad \mathsf{fold}^{\mathsf{CI}}\ K\ R\ (R\ (\mathsf{Fix}^{\mathsf{CI}}\ K\ R))\ (\mathsf{fmap}^{\mathsf{I}}\ K\ R\ (\mathsf{in}^{\mathsf{CI}}\ K\ R))\ w\end{array} \tag{24}$$

3.2 Proof algebras

The impredicative encoding makes it comparatively easy to represent MDTs in Coq, but leaves us with the problem of how to reason inductively about them. Unlike the in-map of the categorical semantics, in^{C} is not a constructor – therefore, structural induction cannot be applied to a term of type $\mathsf{Fix}^{\mathsf{C}}\ F$. Let $P : T \to \mathsf{P}$ be a property and T the representation of an inductive datatype in the following goal, which we assume to be semantically provable by induction on T.

$$\Gamma, w : T \vdash g : P\ w \tag{25}$$

However, given $T =_{df} \mathsf{Fix}^{\mathsf{C}}\ F$ and the impredicative definition of $\mathsf{Fix}^{\mathsf{C}}$, the type T is not syntactically inductive, and no conventional induction principle can be applied. Nevertheless, we can prove

$$\forall v : T.\ \exists w : F\ T.\ P\ v\ =\ P\ (\mathsf{in}^{\mathsf{C}}\ F\ w) \tag{26}$$

as this follows from the equality $v\ =\ \mathsf{in}^{\mathsf{C}}\ F\ (\mathsf{out}^{\mathsf{C}}\ F\ v)$ which can be proved, provided $\mathsf{in}^{\mathsf{C}}\ F$ is shown to be an isomorphism – e.g., by proving (18). Rewriting (25) with (26), we obtain

$$\Gamma, w : F\ T \vdash g' : P\ (\mathsf{in}^{\mathsf{C}}\ F\ w) \tag{27}$$

Here it is possible to apply induction on w, since $F\ T$ is an inductive datatype: however, what we actually get is case analysis – the recursive arguments in $F\ T$ are hidden in the same sense as before, as they have type T rather than $F\ T$.

The solution adopted by Delaware *et al.* in [7], implemented in Coq and supported by MTC/3MT consists of packing an existential copy of the inductive term together with a proof that it satisfies the property, using Σ types. This involves replacing the conventional proof with one based on the representation of the goal as an algebra, i.e., a *proof algebra*.

$$\Gamma \vdash f : \ \mathsf{Alg}^{\mathsf{C}}\ F\ (\Sigma v.\ P\ v) \tag{28}$$

By folding such an algebra, one obtains

$$\Gamma, w : T \vdash \mathsf{fold}^{\mathsf{C}}\ F\ (\Sigma v.\ P\ v)\ f\ w : \ \Sigma v.\ P\ v \tag{29}$$

which states something weaker than the original goal (25). Nonetheless, under conditions associated with *well-formed proof algebras* in [7], (28) can be strengthened to (25). This technique is quite general, and it can be applied to inductive proofs in which the goals may depend on the inductive argument (i.e., it can deal with *dependent induction*). However, the proofs that are obtained in this way are essentially factored into two non-trivial parts – the application of a weak induction principle and a well-formedness proof – and therefore are quite different from conventional inductive ones.

3.3 Looking for a simpler solution

A natural question arises: is it possible to sacrifice some of the generality of the MTC approach, to obtain proofs that look more familiar? The whole point of using Σ types is to hide dependencies: a solution that does not involve them and so a positive answer to our question appear more feasible, when we can dispense with the use of dependent induction, by finding an alternative, equivalent formulation of the goal. In our schematic example (25) we get such reformulation, when we can find $S,\ Q : T \to \mathsf{P}$ and an indexed functor $R : (T \to \mathsf{P}) \to T \to \mathsf{P}$ such that $S =_{df} \mathsf{Fix}^{\mathsf{CI}}\ T\ R$, the following equivalence holds

$$\text{there exists } t \text{ s.t. } \ \Gamma \vdash t : \ \forall w : T.\ S\ w \to Q\ w \qquad \text{iff} \qquad \text{there exists } t' \text{ s.t. } \ \Gamma \vdash t' : \ \forall w : T.\ P\ w \tag{30}$$

and the following is semantically provable, as the new goal, by induction on h:

$$\Gamma,\ w : T,\ h : S\ w \vdash l : Q\ w \tag{31}$$

Intuitively, this means that the dependency of the proof on w can be lifted to a type dependency, given a sufficiently close analogy between T as modular inductive datatype and S as modular inductive predicate, therefore by rather using h of type $S\ w$ as inductive argument. Again, we need to expose the inductive structure by shifting to

$$\Gamma,\ w : T,\ h : R\ (\mathsf{Fix}^{\mathsf{CI}}\ T\ R)\ w \vdash l' : Q\ w \tag{32}$$

and this is not problematic. However, as before, we end up stuck with case analysis rather than proper induction. In order to solve this problem, we need to look at an alternative encoding of fixed points, based on Mendler-style induction [12, 1]. In fact, Mendler's approach makes it possible to build induction principles into impredicatively encoded fixed points. Notice that Mendler algebras are used by Delaware *et al.* [7], but have a different purpose there (i.e., controlling the order of evaluation), from the one we are proposing here.

4 Mendler algebras

We first present the Mendler-style semantics of inductive datatypes by introducing Mendler algebras as a category, following Uustalu and Vene [18]. Given a covariant functor $F : \mathsf{S} \to \mathsf{S}$, a Mendler algebra is a pair $\langle C, \Psi \rangle$ where $C : \mathsf{S}$ is the carrier and $\Psi\ A :\ (A \to C) \to (F\ A \to C)$, for each $A : \mathsf{S}$, is a map from morphisms to morphisms satisfying $\Psi\ A\ f\ =\ (\Psi\ C\ \mathsf{id}_C) \cdot (\mathsf{fmap}\ F\ f)$, with f a morphism from A to C. A morphism between Mendler algebras $\langle C_1, \Psi_1 \rangle$ and $\langle C_2, \Psi_2 \rangle$, is a morphism $h : C_1 \to C_2$ that satisfies $h \cdot \Psi_1\ C_1\ \mathsf{id}_{C_1} = \Psi_2\ C_1\ h$. The Mendler algebra semantics has been proved equivalent to the conventional one by Uustalu *et al.*. Assume F such that the conventional initial F-algebra $\langle \mu F, \mathsf{in}_F \rangle$ exists. Given the abbreviation

$$\mathsf{pre\_in}_F\ C\ (m : C \to \mu F)\ =_{df}\ \mathsf{in}_F \cdot (\mathsf{fmap}\ F\ m) :\ (F\ C \to \mu F) \tag{33}$$

we can prove the equation

$$\mathsf{in}_F\ =\ \mathsf{pre\_in}_F\ \mu F\ \mathsf{id} \tag{34}$$

by the isomorphic character of in_F. The Mendler algebra $\langle \mu F,\ \mathsf{pre\_in}_F \rangle$ can thus be shown to be the initial object in its category, and therefore used as alternative interpretation of the inductive datatype associated with F. For each Mendler algebra $\langle C, \Psi \rangle$, the unique incoming morphism from the initial Mendler F-algebra can be defined

$$\mathsf{mfold}\ F\ C\ \Psi\ x\ =_{df}\ \Psi\ (\mu F)\ (\mathsf{mfold}\ F\ C\ \Psi)\ (\mathsf{out}_F\ x) \tag{35}$$

Unlike the conventional fixpoint operator, the Mendler one can be encoded in Coq as an inductive datatype (though using the impredicative option).

$$\mathsf{dt\_def}\ \mathsf{MFix}\ F\ =\ \mathsf{pre\_in}\ (C : \mathsf{S})\ (b : C \to \mathsf{MFix}\ F)\ (c : F\ C) \tag{36}$$

However in, as defined by equation (34) in this setting, is still not a constructor, and the definition of mfold is not structurally recursive. Therefore, also in this case, it seems more convenient to resort to an impredicative encoding, following [12, 7].

4.1 Impredicative Mendler algebra encoding

Mendler algebras can be characterised impredicatively by the type of their structure maps, and a fixpoint operator can be defined as in the conventional case [12, 7].

$$\mathsf{Alg}^{\mathsf{M}}\ F\ C\ =_{df}\ \forall A.\ (A \to C) \to (F\ A \to C) \tag{37}$$

$$\mathsf{Fix}^{\mathsf{M}}\ F\ =_{df}\ \forall C.\ \mathsf{Alg}^{\mathsf{M}}\ F\ C \to C \tag{38}$$

Unlike the conventional case, the type of a Mendler algebra can be read as specification of an iteration step, where the bound type variable A represents the type of the recursive calls. The corresponding fold operator

$$\mathsf{fold}^{\mathsf{M}}\ F\ C\ f\ x\ =_{df}\ x\ C\ f \tag{39}$$

indeed has type

$$\mathsf{fold}^{\mathsf{M}}\ F\ C :\ (\forall A.\ (A \to C) \to (F\ A \to C)) \to (\mathsf{Fix}^{\mathsf{M}}\ \mathsf{F}) \to C \tag{40}$$

which can represent an induction principle, under the assumption that the argument to the induction hypothesis is only used therein without further analysis [12, 1]. In-maps and out-maps can be defined as follows

$$\mathsf{in}^{\mathsf{M}}\ F\ (x : F(\mathsf{Fix}^{\mathsf{M}}\ F)) : \mathsf{Fix}^{\mathsf{M}}\ F\ =_{df}\ \lambda A\ (f : \mathsf{Alg}^{\mathsf{M}}\ F\ A).\ f\ (\mathsf{Fix}^{\mathsf{M}}\ F)\ (\mathsf{fold}^{\mathsf{M}}\ F\ A\ f)\ x \tag{41}$$

$$\begin{array}{l}\mathsf{out}^{\mathsf{M}}\ F\ (x : \mathsf{Fix}^{\mathsf{M}}\ F) : F\ (\mathsf{Fix}^{\mathsf{M}}\ F)\ =_{df}\ x\ (F\ (\mathsf{Fix}^{\mathsf{M}}\ F)) \\ \qquad (\lambda A\ (r : A \to F\ (\mathsf{Fix}^{\mathsf{M}}\ F))\ (a : F\ A).\ \mathsf{fmap}\ F\ (\lambda y : A.\ \mathsf{in}^{\mathsf{M}}\ F\ (r\ y))\ a)\end{array} \tag{42}$$

As in the conventional case, impredicative fixpoint definitions give us quasi-initiality. The uniqueness condition of $\mathsf{fold}^{\mathsf{M}}\ F\ A\ f$ that is needed for initiality, in a way which parallels (18), is given by

$$(\forall x : F\ (\mathsf{Fix}^{\mathsf{M}}\ F).\ h\ (\mathsf{in}^{\mathsf{M}}\ F\ x)\ =\ f\ (\mathsf{Fix}^{\mathsf{M}}\ F)\ h\ x)\ \to\ h\ =\ \mathsf{fold}^{\mathsf{M}}\ F\ A\ f \tag{43}$$

to be proven for a fixed F, for every $A : \mathsf{S}$, $f : \mathsf{Alg}^{\mathsf{M}}\ F\ A$ and $h : \mathsf{Fix}^{\mathsf{M}}\ F \to A$ [18].

4.2 Indexed Mendler algebras

As before, we need indexed algebras to deal with relations. The definitions are similar to the conventional ones, with K a type, $R : (K \to \mathsf{P}) \to (K \to \mathsf{P})$ an indexed functor, and $D : K \to \mathsf{P}$ an indexed carrier.

$$\mathsf{Alg}^{\mathsf{MI}}\ K\ R\ D =_{df} \forall A.\ (\forall w : K.\ A\ w \to D\ w) \to \forall w : K.\ R\ A\ w \to D\ w \tag{44}$$

$$\mathsf{Fix}^{\mathsf{MI}}\ K\ R\ w =_{df} \forall A.\ \mathsf{Alg}^{\mathsf{MI}}\ K\ R\ A \to A\ w \tag{45}$$

$$\mathsf{fold}^{\mathsf{MI}}\ K\ R\ D\ (f : \mathsf{Alg}^{\mathsf{MI}}\ K\ R\ D)\ (w : K)\ (x : \mathsf{Fix}^{\mathsf{MI}}\ K\ R\ w) =_{df} x\ D\ f \tag{46}$$

$$\begin{array}{l}\mathsf{in}^{\mathsf{MI}}\ K\ R\ (w : K)\ (x : R\ (\mathsf{Fix}^{\mathsf{MI}}\ K\ R)\ w) : \mathsf{Fix}^{\mathsf{MI}}\ K\ R\ w\ =_{df} \\ \qquad \lambda A\ (f : \mathsf{Alg}^{\mathsf{MI}}\ K\ R\ A).\ f\ (\mathsf{Fix}^{\mathsf{MI}}\ K\ R)\ (\mathsf{fold}^{\mathsf{MI}}\ K\ R\ A\ f)\ w\ x\end{array} \tag{47}$$

$$\begin{array}{l}\mathsf{out}^{\mathsf{MI}}\ K\ R\ (w : K)\ (x : \mathsf{Fix}^{\mathsf{MI}}\ K\ R\ w) : R\ (\mathsf{Fix}^{\mathsf{MI}}\ K\ R)\ w\ = \\ \quad x\ (R\ (\mathsf{Fix}^{\mathsf{MI}}\ K\ R))\ (\lambda\ A\ (r : \forall v.\ A\ v \to R\ (\mathsf{Fix}^{\mathsf{MI}}\ K\ R)\ v) \\ \qquad (w : K)\ (a : R\ A\ w).\ \mathsf{fmap}^{\mathsf{I}}\ R\ (\lambda y : A\ w.\ \mathsf{in}^{\mathsf{MI}}\ K\ R\ w\ (r\ w\ y))\ a)\end{array} \tag{48}$$

As an example, we can define inductively a relation $\mathsf{Eval} : (\mathsf{Trm} * \mathsf{Val}) \to \mathsf{P}$ that agrees with eval.

$$\begin{array}{l}\mathsf{dt\_def}\ \mathsf{Eval}_{\mathsf{G}}\ (A : (\mathsf{Trm} * \mathsf{Val}) \to \mathsf{P})\ :\ (\mathsf{Trm} * \mathsf{Val}) \to \mathsf{P}\ = \\ \quad \mathsf{ev1} : \forall x : \mathsf{Int}.\ \mathsf{Eval}_{\mathsf{G}}\ A\ (\mathsf{lit}\ x, \mathsf{val}\ x) \\ \quad \mathsf{ev2} : \forall e_1\ e_2 : \mathsf{Trm}, x_1\ x_2 : \mathsf{Val}.\ A(e_1,x_1)\ \wedge\ A(e_2,x_2) \to \\ \qquad\qquad \mathsf{Eval}_{\mathsf{G}}\ A\ (\mathsf{add}(e_1,e_2), \mathsf{val}((\mathsf{vv}\ x_1) + (\mathsf{vv}\ x_2)))\end{array} \tag{49}$$

$$\mathsf{Eval}\ =_{df}\ \mathsf{Fix}^{\mathsf{MI}}\ (\mathsf{Trm} * \mathsf{Val})\ \mathsf{Eval}_{\mathsf{G}} \tag{50}$$

4.3 Proof algebras, Mendler-style

Reconsider the schematic example in Section 3.2: the problem in (32) was the missing induction hypothesis, that cannot be obtained by appealing to the standard inductive principle, as the recursive occurrences are wrapped in a non-inductive type. Intuitively, this can be fixed by giving such an hypothesis explicitly. This would give us a generic representation of the step lemma in our inductive proof.

$$\Gamma,\ h_0 : \forall v : T.\ \mathsf{Fix}^{\mathsf{CI}}\ T\ R\ v \to Q\ v,\ w : T,\ h_1 : R\ (\mathsf{Fix}^{\mathsf{CI}}\ T\ R)\ w\ \vdash\ q : Q\ w \tag{51}$$

However, here the type of h_0 is actually too specific to be that of the induction hypothesis with respect to h_1 – as a result, the sequent is too weak to take us to the main goal (31). At this point, Mendler's intuition comes into play: under the assumption that the argument passed to the induction hypothesis is used only there, without further case analysis, and that therefore we make no use of its type structure, its type can be represented by a fresh type variable – the key feature of Mendler-style induction [12, 1]. We can then strengthen (51) to the following, more abstract goal.

$$\Gamma,\ A : \mathsf{Type},\ h_0 : \forall v : T.\ A\ v \to Q\ v,\ w : T,\ h_1 : R\ A\ w\ \vdash\ p : Q\ w \tag{52}$$

Given $f =_{df} \lambda A\ h_0\ w\ h_1.\ p$, the above is equivalent to

$$\Gamma\ \vdash\ f :\ \mathsf{Alg}^{\mathsf{MI}}\ T\ R\ Q \tag{53}$$

Now we have an indexed Mendler algebra. The original goal, equivalent to (25) by a reformulation of (30) with $S = \mathsf{Fix}^{\mathsf{MI}}\ T\ R$, can then be obtained by folding, without need of further adjustments.

$$\Gamma\ \vdash\ \mathsf{fold}^{\mathsf{MI}}\ T\ R\ Q\ f\ :\ \forall w : T.\ S\ w \to Q\ w \tag{54}$$

In order to prove (52), case analysis (as provided in Coq e.g. by *inversion* and *destruct* tactics [3]) can be applied to h_1, allowing us to reason on the structure of $R\ A\ w$. This actually results in doing induction on that structure, as the induction hypothesis h_0 is already there. In this way, we can minimise the overhead of combining inductive proofs with modular datatypes. Proving an inductive lemma boils down to constructing the appropriate Mendler algebra – the rest is either conventional, or comes for free. In connection with MDT, such algebras can be regarded as proof modules, that can be composed together in the usual sense of case analysis on coproducts [16, 7], in the same straightforward way as evaluation algebras (the original motivating example by Swierstra [16]). This sounds attractive, from the point of view of the applications in which the relational aspect is predominant, such as structural operational semantics.

4.4 Problematic aspects

Which could be the downsides of the Mendler-based approach? As already observed, relying on impredicative encodings gives us for free only a weak semantics of inductive datatypes, i.e., a quasi-initial one. However, initiality is needed virtually everywhere in our proofs, to ensure in-maps and out-maps are inverses, i.e.

$$(A)\ \ \mathsf{out}^{\mathsf{M}}\ F\ (\mathsf{in}^{\mathsf{M}}\ F\ x)\ =\ x \qquad\qquad (B)\ \ \mathsf{in}^{\mathsf{M}}\ F\ (\mathsf{out}^{\mathsf{M}}\ F\ x)\ =\ x \tag{55}$$

and similarly for the indexed case. In order to get proper initial semantics, functor-specific proofs of properties such as (43) for base category S, or the corresponding one for $K \to \mathsf{P}$, need to be carried out.

This may be regarded as a general weakness of impredicative approaches including MTC/3MT [7, 6], as remarked by Keuchel and Schrijvers [11]. Nonetheless, in discussing the well-formedness of Church encodings [7], Delaware *et al.* argue that dealing with this issue is not too hard, as indeed MTC provides automation for doing so.

A more specific problem is related to the iterative character of Mendler-style recursion, and correspondingly, to the non-dependent character of Mendler-style induction. Mendler algebras make it possible to factor induction into case analysis and folding, but this restricts induction, in the sense of what is called *Mendler iteration* by Abel, Matthes and Uustalu [1]: the argument of the induction hypothesis cannot be used anywhere else, effectively ruling out dependent induction. This means there are problems that cannot be solved in their original form. As an example, MTC [7] proves the type soundness of a language with a dynamic semantics that is recursively defined as a total evaluation function. This problem can be reformulated with respect to our concrete example in Section 2, using our definition of eval (12).

$$\Gamma, e : \mathsf{Trm}, t : \mathsf{Typ} \vdash k : \mathsf{TypOf}\ (e,\ t) \to \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ (\mathsf{eval}\ e),\ t) \tag{56}$$

Using the MTC approach, (56) can be proved by dependent induction on the structure of term e. Given $\mathsf{dt\_def}\ \mathsf{Typ} = \mathsf{N}$ and assuming for simplicity TypOf is a conventional inductive predicate

$$\begin{array}{ll} \mathsf{dt\_def} & \mathsf{TypOf} : \mathsf{Trm} * \mathsf{Typ} \to \mathsf{P} = \\ & \mathsf{tof1} : \forall v : \mathsf{Val}.\ \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ v,\ \mathsf{N}) \\ & \mathsf{tof2} : \forall e_1\ e_2 : \mathsf{Trm}.\ \mathsf{TypOf}\ (e_1, \mathsf{N}) \wedge \mathsf{TypOf}\ (e_2, \mathsf{N}) \to \mathsf{TypOf}\ (\mathsf{add}(e_1, e_2), \mathsf{N}) \end{array} \tag{57}$$

the proof is ultimately based on a proof algebra of type $\mathsf{Alg}^{\mathsf{C}}\ \mathsf{Trm}_{\mathsf{G}}\ (\Sigma e.\ \forall t : \mathsf{Typ}.\ \mathsf{TypOf}\ (e,\ t) \to \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ (\mathsf{eval}\ e),\ t))$, although as already noticed, folding this algebra only gives us the backbone of the whole proof.

This is not possible using our Mendler-style approach, as we cannot deal with the dependency of the goal on the inductive argument e. What we can do instead, is to rely on the relational formulation of evaluation given by Eval (50), which can be shown to satisfy (30), and prove

$$\Gamma, e : \mathsf{Trm}, v : \mathsf{Val},\ t : \mathsf{Typ},\ h : \mathsf{Eval}\ (e, v) \vdash l : \mathsf{TypOf}\ (e,\ t) \to \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ v,\ t) \tag{58}$$

reasoning by induction on the structure of Eval. This reformulation of the goal essentially matches (31). In this case, a proof can be obtained by simply folding an indexed Mendler algebra of type $\mathsf{Alg}^{\mathsf{MI}}\ (\mathsf{Trm} * \mathsf{Val})\ \mathsf{Eval}_{\mathsf{G}}\ (\lambda(e, v).\ \forall t : \mathsf{Typ}.\ \mathsf{TypOf}\ (e,\ t) \to \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ v,\ t))$, which provides our instance of (53).

An alternative way to obtain a relational equivalent of (56) is to lift the modular datatype Trm to a modular predicate $\mathsf{IsTrm} : (\mathsf{Trm}_{\mathsf{G}}\ \mathsf{Trm}) \to \mathsf{P}$, with $\mathsf{IsTrm} =_{df} \mathsf{Fix}^{\mathsf{MI}}\ (\mathsf{Trm}_{\mathsf{G}}\ \mathsf{Trm})\ \mathsf{IsTrm}_{\mathsf{G}}$, where

$$\begin{array}{lll} \mathsf{dt\_def}\ \mathsf{IsTrm}_{\mathsf{G}}\ A & = & \mathsf{isLit} : \forall x : \mathsf{Int}.\ \mathsf{IsTrm}_{\mathsf{G}}\ A\ (\mathsf{lit}\ x) \\ & & \mid \mathsf{isAdd} : \forall e_1\ e_2 : \mathsf{Trm}.\ A\ e_1 \wedge A\ e_2 \to \mathsf{IsTrm}_{\mathsf{G}}\ A\ (\mathsf{add}\ (e_1, e_2)) \end{array} \tag{59}$$

and then prove

$$\Gamma, e : \mathsf{Trm}, w : \mathsf{IsTrm}\ e, t : \mathsf{Typ} \vdash k : \mathsf{TypOf}\ (e,\ t) \to \mathsf{TypOf}\ (\mathsf{lit} \circ \mathsf{vv}\ (\mathsf{eval}\ e),\ t) \tag{60}$$

reasoning by Mendler induction on w. Notice that eval in the MTC example [7] is actually defined as the fold of a Mendler algebra, rather than a conventional one, in order to allow for control over the evaluation order – this is related to the form of their semantics though, and completely unrelated to our use of Mendler-style induction.

5 Case study

The use of relational formulations appears particularly natural in specifications based on small-step rules in the style of SOS, originally introduced by Plotkin [14]. Yet in order to formulate each relation modularly, we need to build encodings based on functors that reflect the structure of those relations. This inevitably makes things more complex, especially when we have to deal with mutually inductive definitions. In order to test the applicability of Mendler proof algebras to the formalisation of a semantic framework, we have formalised a language $\mathscr{L}$ with a comparatively rich syntactic structure, including types (Typ), patterns (Pat), declarations (Dec) and expressions (Exp), as well as value environments ($\mathsf{Env}^{\mathsf{E}}$) and typing environments ($\mathsf{Env}^{\mathsf{T}}$). We rely on SOS to give a partial specification of the language: partial, insofar as we do not specify any behaviour in case of pattern matching failure – therefore, we cannot prove type soundness, which in fact does not hold. However, we can still prove type preservation – and this suffices for us, as an example of the structural complexity we are aiming at.

The full language specification is available with the Coq formalisation in the companion code at [17]. Here we outline the specification using conventional dataytpes. The Coq formalisation is entirely based on modular datatypes, although for simplicity we rely on monolithic functors (we have not yet implemented the smart constructor mechanism that facilitates the use of coproducts).

$$\begin{array}{llll} \mathsf{dt\_def} & \mathsf{Typ} & = & \mathsf{ty}(\mathsf{Id}^{\mathsf{T}}) \mid \mathsf{Typ} \Rightarrow \mathsf{Typ} \mid \mathsf{type\_env}(\mathsf{Env}^{\mathsf{T}}) \\ \mathsf{dt\_def} & \mathsf{Pat} & = & \mathsf{vr}^{\mathsf{p}}(\mathsf{Id},\mathsf{Typ}) \mid \mathsf{cn}^{\mathsf{p}}(\mathsf{Id},\mathsf{Typ}) \mid \mathsf{apply}^{\mathsf{p}}(\mathsf{Pat},\mathsf{Pat}) \\ \mathsf{dt\_def} & \mathsf{Dec} & = & \mathsf{env}(\mathsf{Env}^{\mathsf{E}}) \mid \mathsf{match}(\mathsf{Pat},\mathsf{Exp}) \mid \mathsf{join}(\mathsf{Dec},\mathsf{Dec}) \\ \mathsf{dt\_def} & \mathsf{Exp} & = & \mathsf{vr}(\mathsf{Id}) \mid \mathsf{cn}(\mathsf{Id},\mathsf{Typ}) \mid \mathsf{closure}(\mathsf{Env}^{\mathsf{E}},\mathsf{Pat},\mathsf{Exp}) \\ & & & \mid \mathsf{apply}(\mathsf{Exp},\mathsf{Exp}) \mid \mathsf{scope}(\mathsf{Dec},\mathsf{Exp}) \end{array} \tag{61}$$

$$\mathsf{Env}\ A =_{df} \mathsf{Id} \to \mathsf{option}\ A \qquad \mathsf{Env}^{\mathsf{T}} =_{df} \mathsf{Env}\ \mathsf{Typ} \qquad \mathsf{Env}^{\mathsf{E}} =_{df} \mathsf{Env}\ \mathsf{Exp} \tag{62}$$

The language $\mathscr{L}$ is based on simply typed lambda calculus with pattern matching and first class environments. We use two sets of identifiers – Id^{T} for type variables and Id for object variables and constants. Constants and pattern variables are annotated with types. $\Rightarrow$ is the usual function type constructor. We use closures instead of lambda abstractions to ensure values are closed terms and avoid dealing with substitution. Abstraction is defined over patterns (rather than simply over variables). Matching patterns with expressions give declarations, which may evaluate to environments. Declarations can be joined together and used in scope expressions. Values can be specified as follows.

$$\begin{array}{lll} \text{Data values :} & h \in & \mathsf{cn}(x,\tau) \mid \mathsf{apply}(h,v) \\ \text{Values :} & v \in & \mathsf{closure}(\rho,p,e) \mid h \end{array} \tag{63}$$

The typing relations have the following signatures. Notice that patterns and values can be typed in a context-free way, unlike expressions and declarations.

$$\begin{array}{ll} \text{Patterns :} & \mathsf{TypOPat} : \mathsf{Pat} * \mathsf{Typ} \to \mathsf{P} \\ \text{Environments :} & \mathsf{TypOEnv} : \mathsf{Env}^{\mathsf{E}} * \mathsf{Env}^{\mathsf{T}} \to \mathsf{P} \\ \text{Declarations :} & \mathsf{TypODec} : \mathsf{Env}^{\mathsf{T}} * \mathsf{Dec} * \mathsf{Typ} \to \mathsf{P} \\ \text{Expressions :} & \mathsf{TypOExp} : \mathsf{Env}^{\mathsf{T}} * \mathsf{Exp} * \mathsf{Typ} \to \mathsf{P} \end{array} \tag{64}$$

The transition relations have the following signatures.

$$\begin{array}{ll} \text{Declarations :} & \mathsf{DecStep} : \mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec} \to \mathsf{P} \\ \text{Expressions :} & \mathsf{ExpStep} : \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp} \to \mathsf{P} \end{array} \tag{65}$$

Expressions and declarations may depend on each other, and therefore can only have a mutually inductive definition. Analogously, the definitions of the typing relations and of the transition relations for these two syntactic categories involve mutual induction. Therefore we need to introduce functors to reason about mutually inductively defined sets, as well as mutually inductively defined relations.

5.1 Mutually inductive sets

Two mutually recursive datatypes in the base category S, can be represented in terms of bi-functors $F_1,\ F_2 :\ \mathsf{S} * \mathsf{S} \to \mathsf{S}$, where bi-functoriality is expressed as existence of a map $\mathsf{fmap}^{\mathsf{D}}$ which satisfies the appropriate form of the usual preservation properties.

$$\mathsf{fmap}^{\mathsf{D}} :\ \forall\ \{A_1\ A_2\ B_1\ B_2 : \mathsf{S}\}\ (f_1 : A_1 \to B_1)\ (f_2 : A_2 \to B_2).\ F\ (A_1,A_2) \to F\ (B_1,B_2) \tag{66}$$

$$\begin{array}{l} \mathsf{fmap}^{\mathsf{D}}\ g_1\ g_2\ (\mathsf{fmap}^{\mathsf{D}}\ f_1\ f_2)\ =\ \mathsf{fmap}^{\mathsf{D}}\ (g_1 \cdot f_1)\ (g_2 \cdot f_2) \\ \mathsf{fmap}^{\mathsf{D}}\ \mathsf{id}_A\ \mathsf{id}_B\ =\ \mathsf{id}_{FAB} \end{array} \tag{67}$$

The definitions of Mendler bi-algebra, fixpoint and fold operators can be given using pairs.

$$\begin{array}{l} \mathsf{Alg}^{\mathsf{D}}\ (F_1,F_2)\ (C_1,C_2)\ =_{df}\ (\forall A_1\ A_2.\ (A_1 \to C_1) \to (A_2 \to C_2) \to F_1\ (A_1,A_2) \to C_1, \\ \qquad\qquad \forall A_1\ A_2.\ (A_1 \to C_1) \to (A_2 \to C_2) \to F_2\ (A_1,A_2) \to C_2) \end{array} \tag{68}$$

$$\begin{array}{l} \mathsf{Fix}^{\mathsf{D}}\ (F_1,F_2)\ =_{df}\ (\forall A_1\ A_2.\ \mathsf{Alg}^{\mathsf{D}}\ (F_1,F_2)\ (A_1,A_2) \to A_1, \\ \qquad\qquad \forall A_1\ A_2.\ \mathsf{Alg}^{\mathsf{D}}\ (F_1,F_2)\ (A_1,A_2) \to A_2) \end{array} \tag{69}$$

$$\begin{array}{l} \mathsf{fold}_1^{\mathsf{D}}\ (F_1,F_2)\ (C_1,C_2)\ (f : \mathsf{Alg}^{\mathsf{D}}\ (F_1,F_2)\ (C_1,C_2)) : \\ \qquad\qquad \mathsf{fst}\ (\mathsf{Fix}^{\mathsf{D}}\ (F_1,F_2)) \to C_1\ =_{df}\ \lambda e.\ e\ C_1\ C_2\ f \end{array} \tag{70}$$

$$\begin{array}{l} \mathsf{fold}_2^{\mathsf{D}}\ (F_1,F_2)\ (C_1,C_2)\ (f : \mathsf{Alg}^{\mathsf{D}}\ (F_1,F_2)\ (C_1,C_2)) : \\ \qquad\qquad \mathsf{snd}\ (\mathsf{Fix}^{\mathsf{D}}\ (F_1,F_2)) \to C_2\ =_{df}\ \lambda e.\ e\ C_1\ C_2\ f \end{array} \tag{71}$$

All the syntactic categories of $\mathscr{L}$ can then be represented as MDTs, using bi-functors for mutually defined Decl and Exp.

$$\begin{array}{l} \mathsf{dt\_def}\ \mathsf{Typ}_{\mathsf{G}}\ T\ =\ \mathsf{ty}(\mathsf{Id}^{\mathsf{T}}) \mid T \Rightarrow T \mid \mathsf{type\_env}\ (\mathsf{Env}^{\mathsf{T}}\ T) \qquad\qquad \mathsf{Typ}\ =_{df}\ \mathsf{Fix}^{\mathsf{M}}\ \mathsf{Typ}_{\mathsf{G}} \\ \mathsf{dt\_def}\ \mathsf{Pat}_{\mathsf{G}}\ P\ =\ \mathsf{vr}^{\mathsf{P}}(\mathsf{Id},T) \mid \mathsf{cn}^{\mathsf{P}}(\mathsf{Id},T) \mid \mathsf{apply}^{\mathsf{P}}(P,P) \qquad\qquad \mathsf{Pat}\ =_{df}\ \mathsf{Fix}^{\mathsf{M}}\ \mathsf{Pat}_{\mathsf{G}} \\ \mathsf{dt\_def}\ \mathsf{Dec}_{\mathsf{G}}\ D\ E\ =\ \mathsf{env}(\mathsf{Env}\ E) \mid \mathsf{match}(\mathsf{Pat},E) \mid \mathsf{join}(D,D) \\ \mathsf{dt\_def}\ \mathsf{Exp}_{\mathsf{G}}\ D\ E\ =\ \mathsf{vr}(\mathsf{Id}) \mid \mathsf{cn}(\mathsf{Id},\mathsf{Typ}) \mid \mathsf{closure}(\mathsf{Env}\ E,\mathsf{Pat},E)\ \mid \mathsf{apply}(E,E) \mid \mathsf{scope}(D,E) \\ \qquad \mathsf{Dec}\ =_{df}\ \mathsf{fst}\ (\mathsf{Fix}^{\mathsf{D}}\ (\mathsf{Dec}_{\mathsf{G}},\mathsf{Exp}_{\mathsf{G}})) \qquad\qquad \mathsf{Exp}\ =_{df}\ \mathsf{snd}\ (\mathsf{Fix}^{\mathsf{D}}\ (\mathsf{Dec}_{\mathsf{G}},\mathsf{Exp}_{\mathsf{G}})) \end{array} \tag{72}$$

5.2 Mutually inductive relations

Given types K_1, K_2, two mutually recursive relations depending on such types in base categories $K_1 \to \mathsf{P}$, $K_2 \to \mathsf{P}$, can be represented by indexed bi-functors R_1, R_2, with

$$R_1\ K_1 : (K_1 \to \mathsf{P}) * (K_2 \to \mathsf{P}) \to (K_1 \to \mathsf{P}) \qquad R_2\ K_1 : (K_1 \to \mathsf{P}) * (K_2 \to \mathsf{P}) \to (K_2 \to \mathsf{P}) \tag{73}$$

characterised by maps

$$\begin{aligned}&\mathsf{fmap}_1^{\mathsf{H}}\ (K_1,K_2)\ R_1 :\ \forall\ \{A_1\ A_2 :\ K_1 \to \mathsf{P}\}\ \{B_1\ B_2 :\ K_2 \to \mathsf{P}\}.\\ &\quad (\forall w : K_1.\ A_1\ w \to B_1\ w) \to (\forall w : K_2.\ A_2\ w \to B_2\ w) \to\\ &\quad\quad \forall w : K_1.\ R_1\ (A_1,A_2)\ w \to R_1\ (B_1,B_2)\ w\end{aligned} \tag{74}$$

$$\begin{aligned}&\mathsf{fmap}_2^{\mathsf{H}}\ (K_1,K_2)\ R_2 :\ \forall\ \{A_1\ A_2 :\ K_1 \to \mathsf{P}\}\ \{B_1\ B_2 :\ K_2 \to \mathsf{P}\}.\\ &\quad (\forall w : K_1.\ A_1\ w \to B_1\ w) \to (\forall w : K_2.\ A_2\ w \to B_2\ w) \to\\ &\quad\quad \forall w : K_2.\ R_2\ (A_1,A_2)\ w \to R_2\ (B_1,B_2)\ w\end{aligned} \tag{75}$$

Given carriers $D_1 : K_1 \to \mathsf{P}$, $D_2 : K_2 \to \mathsf{P}$, we can now define indexed Mendler bi-algebras and the associated notions (see [17] for more details).

$$\begin{aligned}&\mathsf{Alg}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (D_1,D_2)\ =_{df}\\ &\quad (\forall A_1\ A_2.\ (\forall w : K_1.\ A_1\ w \to D_1\ w) \to (\forall w : K_2.\ A_2\ w \to D_2\ w) \to\\ &\quad\quad \forall w : K_1.\ R_1\ (A_1,A_2)\ w \to D_1\ w,\\ &\quad \forall A_1\ A_2.\ (\forall w : K_1.\ A_1\ w \to D_1\ w) \to (\forall w : K_2.\ A_2\ w \to D_2\ w) \to\\ &\quad\quad \forall w : K_2.\ R_2\ (A_1,A_2)\ w \to D_2\ w)\end{aligned} \tag{76}$$

$$\begin{aligned}&\mathsf{Fix}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ =_{df}\\ &\quad (\lambda w : K_1.\ \forall A_1\ A_2.\ \mathsf{Alg}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (A_1,A_2) \to A_1\ w,\\ &\quad \lambda w : K_2.\ \forall A_1\ A_2.\ \mathsf{Alg}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (A_1,A_2) \to A_2\ w)\end{aligned} \tag{77}$$

$$\begin{aligned}&\mathsf{fold}_1^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (D_1,D_2)\ (f : \mathsf{Alg}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (D_1,D_2))\ (w : K_1) :\\ &\quad \mathsf{fst}\ (\mathsf{Fix}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2))\ w \to D_1\ w\ =_{df}\ \lambda w\ e.\ e\ D_1\ D_2\ f\end{aligned} \tag{78}$$

$$\begin{aligned}&\mathsf{fold}_2^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (D_1,D_2)\ (f : \mathsf{Alg}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2)\ (D_1,D_2))\ (w : K_2) :\\ &\quad \mathsf{snd}\ (\mathsf{Fix}^{\mathsf{H}}\ (K_1,K_2)\ (R_1,R_2))\ w \to D_2\ w\ =_{df}\ \lambda w\ e.\ e\ D_1\ D_2\ f\end{aligned} \tag{79}$$

While the typing relations for patterns TypOPat can be represented modularly using an indexed functor and $\mathsf{Fix}^{\mathsf{I}}$, the corresponding relations for declarations and expressions, i.e. TypODec and TypOExp respectively, are mutually defined and therefore need to be represented as indexed bi-functors closed by $\mathsf{Fix}^{\mathsf{H}}$. Such is also the case for DecStep and ExpStep, which can be defined as follows, given the corresponding indexed bi-functors $\mathsf{DecStep}_{\mathsf{G}} : (\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec} \to \mathsf{P},\ \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp} \to \mathsf{P}) \to \mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec} \to \mathsf{P}$, and $\mathsf{ExpStep}_{\mathsf{G}} : (\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec} \to \mathsf{P},\ \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp} \to \mathsf{P}) \to \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp} \to \mathsf{P}$.

$$\mathsf{DecStep}\ =_{df}\ \mathsf{fst}\ (\mathsf{Fix}^{\mathsf{H}}\ (\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec},\ \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp})\ (\mathsf{DecStep}_{\mathsf{G}},\ \mathsf{ExpStep}_{\mathsf{G}})) \tag{80}$$

$$\mathsf{ExpStep}\ =_{df}\ \mathsf{snd}\ (\mathsf{Fix}^{\mathsf{H}}\ (\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec},\ \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp})\ (\mathsf{DecStep}_{\mathsf{G}},\ \mathsf{ExpStep}_{\mathsf{G}})) \tag{81}$$

5.3 Type preservation

Type preservation in $\mathscr{L}$ can be expressed as follows

$$\begin{aligned}&\Gamma, \rho : \mathsf{Env}^{\mathsf{E}}\ \vdash\ (\forall (d_1\ d_2 : \mathsf{Dec}).\ \mathsf{DecStep}\ (\rho, d_1, d_2) \to \mathsf{DecTSafe}\ (\rho, d_1, d_2)\\ &\quad\quad \wedge\ (\forall (e_1\ e_2 : \mathsf{Exp}).\ \mathsf{ExpStep}\ (\rho, e_1, e_2) \to \mathsf{ExpTSafe}\ (\rho, e_1, e_2)\end{aligned} \tag{82}$$

where

$$\begin{array}{l}\mathsf{DecTSafe}\ (\rho,d_1,d_2) =_{df}\ \forall(t:\mathsf{Typ})\ (\gamma:\mathsf{Env}^{\mathsf{T}}).\\ \qquad \mathsf{TypOEnv}\ (\rho,\gamma) \to \mathsf{TypODec}\ (\gamma,d_1,t) \to \mathsf{TypODec}\ (\gamma,d_2,t)\\ \mathsf{ExpTSafe}\ (\rho,e_1,e_2) =_{df}\ \forall(t:\mathsf{Typ})\ (\gamma:\mathsf{Env}^{\mathsf{T}}).\\ \qquad \mathsf{TypOEnv}\ (\rho,\gamma) \to \mathsf{TypOExp}\ (\gamma,e_1,t) \to \mathsf{TypOExp}\ (\gamma,e_2,t)\end{array} \tag{83}$$

The context Γ includes premises of shape

$$(IN\,x\ =\ IN\,y)\ \to\ (x\ =\ y) \tag{84}$$

where IN is the in-map for one of the datatypes – such premises can be discharged when the corresponding initiality conditions (43) are proven. It also includes premises of shape

$$\forall x : D_G, IsD_G\,x. \tag{85}$$

where D_G is the unfolding of a modular datatype D, and IsD_G is the unfolding of a modular predicate IsD that represents the relational lifting of D, in the sense of our example (59). Such premises are needed, as the proof involves sublemmas that are proved by induction on the syntactic categories – and so, for instance, $\mathsf{Typ}_{\mathsf{G}}\ \mathsf{Typ}$ has to be lifted to $\mathsf{IsTyp}_{\mathsf{G}} : (\mathsf{Typ}_{\mathsf{G}}\ \mathsf{Typ} \to \mathsf{P}) \to \mathsf{Typ}_{\mathsf{G}}\ \mathsf{Typ} \to \mathsf{P}$.

Crucially, the pair of DecTSafe and ExpTSafe can be a carrier for the indexed bi-functor determined by DecStep and ExpStep. In order to prove type preservation by mutual induction on the structure of DecStep and ExpStep, we define an indexed Mendler bi-algebra that has $(\mathsf{DecTSafe}, \mathsf{ExpTSafe})$ as indexed carrier, where the index types are $\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec}$ and $\mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp}$

$$\begin{array}{l}\mathsf{TPAlg}\ =_{df}\ \mathsf{Alg}^{\mathsf{H}}\ (\mathsf{Env}^{\mathsf{E}} * \mathsf{Dec} * \mathsf{Dec},\ \mathsf{Env}^{\mathsf{E}} * \mathsf{Exp} * \mathsf{Exp})\\ \qquad\qquad (\mathsf{DecStep}_{\mathsf{G}},\ \mathsf{ExpStep}_{\mathsf{G}})\ (\mathsf{DecTSafe},\ \mathsf{ExpTSafe})\end{array} \tag{86}$$

After finding proofs $f_1 : \mathsf{fst}\ \mathsf{TPAlg}$ and $f_2 : \mathsf{snd}\ \mathsf{TPAlg}$, we can construct a proof of (82) by applying to them $\mathsf{fold}_1^{\mathsf{H}}$ and $\mathsf{fold}_2^{\mathsf{H}}$, respectively (see [17] for details).

6 Conclusion

Motivated by the importance of modularity in program development, semantics and verification, we have discussed the use of MDTs, their semantic foundations and their impredicative encoding along the lines of existing work [7, 11, 16]. We have shown how impredicative MDT encodings based on Mendler algebras can be used to reason about inductively defined relations, in a way that is comparatively close to a more conventional style of reasoning based on closed datatypes, by providing a simpler notion of proof algebra, if less general, than the one proposed by Delaware *et al.* [7]. Our approach can be regarded as a novel application of Mendler-style induction [12, 1, 18], as well as a technique that could be integrated in existing frameworks based on the impredicative encoding, such as MTC/3MT [7, 6]. Mendler's original insight [12] was in the semantics of inductive datatypes – the case made here, is for using that insight as a modular proof technique. From the point of view of possible applications to semantics and verification in frameworks such as OTT [15], the relational style that can be supported seems to fit in well with SOS and in particular with component-based approaches, such as the one proposed by Churchill, Mosses, Sculthorpe and Torrini [4]. Our plans for future work include integrating our technique in MTC/3MT, and comparing this approach with the container-based one proposed by Keuchel and Schrijvers [11].

Acknowledgments: We thank Steven Keuchel, Neil Sculthorpe, Casper Bach Poulsen and the anonymous reviewers for feedback on earlier versions, and members of the Theory Group at Swansea University, including Peter Mosses, Anton Setzer and Ulrich Berger, for discussion. The writing of this paper has been supported by EU funding (H2020 FET) to KU Leuven for the GRACEFUL project. Preliminary work was funded by the EPSRC grant (EP/I032495/1) to Swansea University for the PLANCOMPS project.

References

[1] A. Abel, R. Matthes & T. Uustalu (2005): *Iteration and coiteration schemes for higher-order and nested datatypes. Theor. Comput. Sci.* 333(1-2), pp. 3–66, doi:10.1016/j.tcs.2004.10.017.

[2] H. Barendregt (1992): *Lambda Calculi with Types*. In: *Hand. of Logic in Co. Sc.*, Oxford, pp. 117–309.

[3] Y. Bertot & P. Casteran (2004): *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, doi:10.1007/978-3-662-07964-5.

[4] M. Churchill, P. D. Mosses, N. Sculthorpe & P. Torrini (2015): *Reusable Components of Semantic Specifications*. In: *TAOSD 12*, LNCS 8989, Springer, pp. 132–179, doi:10.1007/978-3-662-46734-3_4.

[5] T. Coquand & G. Huet (1988): *The calculus of constructions*. *Information and Computation* 76, pp. 95–120, doi:10.1016/0890-5401(88)90005-3.

[6] B. Delaware, S. Keuchel, T. Schrijvers & B. C.d.S. Oliveira (2013): *Modular Monadic Meta-theory*. In: *ICFP'13*, ACM, pp. 319–330, doi:10.1145/2500365.2500587.

[7] B. Delaware, B. C. d. S. Oliveira & T. Schrijvers (2013): *Meta-theory à la carte*. In: *Proc. POPL '13*, pp. 207–218, doi:10.1145/2429069.2429094.

[8] N. Ghani, T. Uustalu & V. Vene (2004): *Build, Augment and Destroy, Universally*. In: *Proc. APLAS '04*, pp. 327–347, doi:10.1007/978-3-540-30477-7_22.

[9] T. Hagino (1987): *A Typed Lambda Calculus with Categorical Type Constructors*. In: *Category Theory and Computer Science*, pp. 140–157, doi:10.1007/3-540-18508-9_24.

[10] G. Hutton (1999): *A Tutorial on the Universality and Expressiveness of Fold*. *J. Funct. Program.* 9(4), pp. 355–372, doi:10.1017/S0956796899003500.

[11] S. Keuchel & T. Schrijvers (2013): *Generic Datatypes à la Carte*. In: *9th ACM SIGPLAN Workshop on Generic Programming (WGP)*, pp. 1–11, doi:10.1145/2502488.2502491.

[12] N. P. Mendler (1991): *Inductive Types and Type Constraints in the Second-Order lambda Calculus*. *Ann. Pure Appl. Logic* 51(1-2), pp. 159–172, doi:10.1016/0168-0072(91)90069-X.

[13] F. Pfenning & C. Paulin-Mohring (1989): *Inductively Defined Types in the Calculus of Constructions*. In: *Math. Foundations of Programming Semantics*, pp. 209–228, doi:10.1007/BFb0040259.

[14] G. D. Plotkin (2004): *A structural approach to operational semantics*. *J. Log. Algebr. Program.* 60-61, pp. 17–139, doi:10.1016/j.jlap.2004.03.009.

[15] P. Sewell, F. Zappa Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar & R. Strniša (2010): *Ott: Effective Tool Support for the Working Semanticist*. *Journal of Functional Programming* 20(1), pp. 71–122, doi:10.1017/S0956796809990293.

[16] W. Swierstra (2008): *Data types à la carte*. *Journal of Functional Programming* 18(4), pp. 423–436, doi:10.1017/S0956796808006758.

[17] P. Torrini (2015): *Language specification and type preservation proofs in Coq – companion code*. Available at `http://cs.swan.ac.uk/~cspt/MDTC`.

[18] T. Uustalu & V. Vene (1999): *Mendler-Style Inductive Types, Categorically*. *Nord. J. Comput.* 6(3), p. 343.

[19] P. Wadler (1990): *Recursive types for free!* Available at `http://homepages.inf.ed.ac.uk/wadler/papers/free-rectypes/free-rectypes.txt`.