A model for behavioural properties of higher-order programs

Sylvain Salvati, Igor Walukiewicz

A model for behavioural properties of higher-order programs

Sylvain Salvati[1] and Igor Walukiewicz[2]

1 **Université de Bordeaux, INRIA, LaBRI UMR5800**
2 **Université de Bordeaux, CNRS, LaBRI UMR5800**

Abstract

We consider simply typed lambda-calculus with fixpoints as a non-interpreted functional programming language: the result of the execution of a program is its normal form that can be seen as a potentially infinite tree of calls to built-in operations. Properties of such trees are properties of executions of programs and monadic second-order logic (MSOL) is well suited to express them.

For a given MSOL property we show how to construct a finitary model recognizing it. In other words, the value of a lambda-term in the model determines if the tree that is the result of the execution of the term satisfies the property. The finiteness of the construction has as consequences many known results about the verification of higher-order programs in this framework.

1 Introduction

Higher-order functions are being adopted by most mainstream programming languages. Higher-order functions not only increase modularity and elegance of the code, but also help to address such fundamental issues as scalability and fault-tolerance. In consequence, higher-order functions are increasingly used for writing programs interacting with an environment, like, for example, client-server web applications. To accompany this evolution, new kinds of analysis tools are needed, focusing on behavioural properties of higher-order functional programs. For example, some guidelines for secure web programming may require that if a database access is required infinitely often then calls to a logging function must be made again and again. Our objective is to develop denotational models for such kinds of properties.

We consider λY-calculus, the simply typed λ-calculus with fixpoints, as an abstraction of a higher-order programming language that faithfully represents the control flow. Under the name of recursive program schemes the calculus has been studied since 1960s [11, 4, 6, 7, 13, 19]. The particularity of this approach is to focus on the free interpretation: all constants are non-interpreted symbols and the interpretation of a term is a tree composed from constants. In the context of λ-calculus this tree is called the Böhm tree. Figure 1 presents the Böhm tree of `map` function. It is a generic iterator taking a function and a list, and applying the function to every element of the list. Observe that even for such a simple program its Böhm tree is infinite and not regular.

Program properties can be grouped in two families. The first, and the most obvious one, concerns the absence of run-time errors. A slogan "well-typed programs never go wrong" clearly expresses this idea. More generally, this family contains all kinds of *safety properties*, i.e., those determined by a set of finite executions considered as admissible. The other family is that of *liveness properties* that talk about infinite executions. For example: "logging function is called again and again" or "every initiated communication is eventually closed". Concerning the `map` function, we can say for example that if l is a finite list then the call $\texttt{map}(f, l)$ evokes f only finitely many times. In fact all fairness properties are particular liveness properties. Such properties are of relevance to servers, web services, operating systems, and more generally, to all kinds of interactive applications. Regarding liveness properties, monadic second-order logic (MSOL), or equivalently automata on infinite

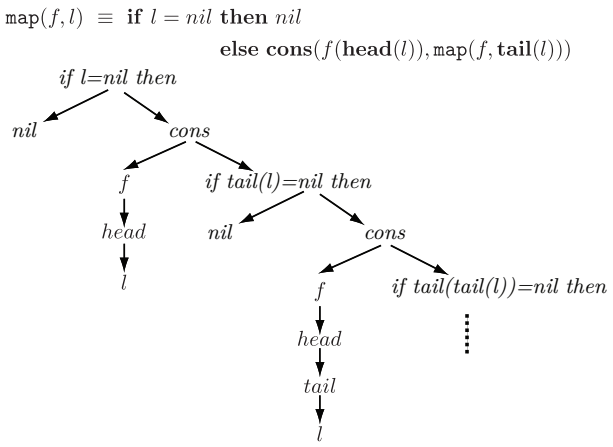

Figure 1 The map function and its semantics in the form of a (simplified) Böhm tree.

objects, sets the standard of expressivity and algorithmic manageability. Moreover, thanks to the result of Ong [19], it is decidable if the Böhm tree of a given term of the λY-calculus satisfies a given MSOL property.

In this paper we show how to construct for a given MSOL property a finitary model so that the value of a term in the model determines if the Böhm tree of the term satisfies the property. More precisely, we work with the formalism of parity automata instead of MSOL. We show that the value of a term of the base type in the model constructed from a given automaton is simply the set of states from which the automaton accepts the Böhm tree of the term (Theorem 11).

Our model construction shows how to extend Scott models to integrate the parity condition of a given automaton. Finitary Scott models are the simplest models of the λY-calculus: the base type is interpreted as a finite lattice, functional types as the sets of monotone functions, and the fixpoint as the least fixpoint. Such models correspond in a precise sense to safety properties, or equivalently to finite automata on infinite trees that are Ω-blind and have trivial acceptance conditions [22]. This implies that in order to capture the expressive power of parity automata some modification of Scott models is needed. The straightforward idea of introducing ranks of the parity condition directly in the base type does not seem to work. Instead our construction introduces ranks only in higher types. The other crucial point is the interpretation of function spaces: we cannot take all monotone functions but only those that behave well with respect to ranks. This is formalized with a new domain identity we call *stratification*.

The model construction gives a completely compositional approach to verification: the result of a term is calculated from the results for its subterms. In particular, we give the meaning of a fixpoint constant as a particular fixpoint of its argument. The construction implies the transfer theorem for MSOL [21], and with it a number of consequences offered by this theorem. Finitary models are used in program transformations: during its execution the program can calculate the values of chosen subterms [22, 9]. In our case it can, for example, detect if an argument satisfies a particular liveness property.

Our construction is based on the insights from a very influential paper of Kobayashi and Ong [16], where, amongst other contributions, they give a typing system to capture the same dependencies inside terms that we represent in our model. Although the quest for models for behavioural properties has began some time ago, the results started to appear only recently. Tsukada and Ong [25] extended the approach from [16] to a typing system for

the whole λY-calculus. In this system the fixpoint is still treated externally via games, and the model underlying the system is not finitary. They use game semantics to understand a difficult problem of the behaviour of the application operation on the level of Böhm trees. Also last year, Hofmann and Chen provided a model for verifying path properties expressed in MSOL [10]. Their construction is restricted only to first-order λY-terms. They use in an elegant way Wilke algebras that are an algebraic notion of recognizer for languages of infinite words. One of the problems we are facing here is that there does not exist equally satisfying notion of an algebraic recognizer for infinite trees. Even if we wanted to stay with properties of paths, it is not clear how to extend Wilke algebras to higher orders, the problem being to find an admissible class of fixpoint operations. More recently, Grellois and Mellies [8] have given a categorical account of the behaviour of ranks in a model. They derive an infinite model via elegant general constructions. About the same time, we have provided a model construction for properties expressed in weak MSOL [23]. The model is a sort of layered Scott model. The restriction to weak MSOL greatly simplifies the integration of ranks in the model. As a consequence, it was possible to adapt classical arguments from domain theory to prove the correctness of the model. The present construction does not follow the line of [23]. Apart from [16], the main influence comes from the work of Mellies [17] clearly showing the value of using the morphism composition similar to that in Kleisli categories. Furthermore, the stratification property is essential to get the model to satisfy the required equations. The proof methods for the correctness of the model are extensions of game based methods we have developed for the proof of the transfer theorem [21].

Apart from model based approaches cited above, there is a very active research in verification of behavioural properties of higher-order programs. Among the closest methods using the class of properties and programs we consider here we can list [15, 3, 20]. Similar research objectives are also pursued in different settings. We would like to mention the work of Naik and Palsberg [18] who make a connection between model-checking and typing. They consider only safety properties, and focus on first-order imperative programs. Another interesting line of research is proposed by Jeffrey [12] who shows how to incorporate Linear Temporal Logic into types using a richer dependent types paradigm. The calculus is intended to talk about control and data in functional reactive programming framework, and aims at using SMT solvers.

Organization of the paper: In the next preliminary section we introduce basic definitions, and present two special cases that allow us to introduce the main concepts in a simpler setting. Section 3 is devoted to the definition of the model and its properties. The main theorem of the paper is stated in this section. The next section presents the proof outline. Section 4 shows some consequences of the model construction. The conclusions section outlines some directions for further research.

2 Preliminaries

We start by introducing λY-calculus and parity automata. Then we present two simple special cases of the main result of the paper. The first case shows what can be achieved with the classical notion of a model for λY-calculus. The second considers only terms of order at most 1. It allows us to introduce some crucial elements of the general solution.

2.1 λY-calculus

The *set of types* is constructed from a unique *basic type* o using a binary operation $\to$ that associates to the right. Thus o is a type and if A, B are types, so is $(A \to B)$. The order of a type is defined by: $order(o) = 0$, and $order(A \to B) = max(1 + order(A), order(B))$. We work with *tree signatures* that are finite sets of *typed constants of order at most* 1. Types of order 1 are of the form $o \to \cdots \to o \to o$ that we abbreviate $o^i \to o$ when they contain $i+1$ occurrences of o. For convenience we assume that $o^0 \to o$ is just o. If Σ is a signature, we write $\Sigma^{(i)}$ for the set of constants of type $o^i \to o$.

Simply typed λY-terms are built from the constants in the signature, and constants Y^A, Ω^A for every type A. These stand for the *fixpoint combinator* and *undefined term*, respectively. Apart from constants, for each type A there is a countable set of variables $x^A, y^A, \ldots$. Terms are built from these constants and variables using typed application and λ-abstraction. We shall write sequences of λ-abstractions $\lambda x_1 \ldots \lambda x_n.\ M$ with only one λ: either $\lambda x_1 \ldots x_n.\ M$, or even shorter $\lambda \vec{x}.\ M$. We take for granted the operational semantics of the calculus given by β and δ reductions.

The *Böhm tree* of a term M is obtained by reducing it until one reaches a term of the form $\lambda \vec{x}.N_0 N_1 \ldots N_k$ with N_0 a variable or a constant. Then $BT(M)$ is a tree having its root labelled by $\lambda \vec{x}.N_0$ and having $BT(N_1), \ldots, BT(N_k)$ as subtrees. Otherwise $BT(M) = \Omega^A$, where A is the type of M. Böhm trees are infinite normal forms of λY-terms. A Böhm tree of a closed term of type o over a tree signature is a potentially infinite ranked tree: a node labelled by a constant a of type $o^i \to o$ has i successors. Among constants Ω^A, only constant Ω^o can appear in the Böhm tree of such a term.

2.2 MSOL and parity automata

We are interested in properties of trees expressed in monadic second-order logic (MSOL). This is an extension of first-order logic with quantification over sets of elements. Over infinite trees MSOL formulas define precisely regular tree languages. This class of languages has numerous other characterizations. Here we will rely on the one using parity tree automata.

Automata will work on Σ-labelled trees, where Σ is a tree signature. Trees are partial functions $t : \mathbb{N}^* \stackrel{\cdot}{\to} \Sigma \cup \{\Omega\}$ such that the number of successors of a node is determined by the label of the node. In particular, if $t(u) \in \Sigma^{(0)}$ then u is a leaf. The *nodes* of t, are the elements of the domain of t. The set of nodes should be prefix closed. A *label* of a node u is $t(u)$.

We will use nondeterministic max-parity automata, that we will call *parity automata* for short. Such an automaton accepts trees over a fixed tree signature Σ. It is a tuple

$$\mathcal{A} = \langle Q, \Sigma, \{\delta_i\}_{i \in \mathbb{N}}, rk : Q \to [m] \rangle$$

where Q is a finite set of states, rk is the *rank function* with the range $[m] = \{0, \ldots, m\}$, and $\delta_i : Q \times \Sigma^{(i)} \to \mathcal{P}(Q^i)$ is the *transition function*. Observe that since the signature Σ is finite, only finitely many δ_i are nontrivial. From the definition it follows that, for example, $\delta_2 : Q \times \Sigma^{(2)} \to \mathcal{P}(Q \times Q)$ and $\delta_0 : Q \times \Sigma^{(0)} \to \{\emptyset, \{\emptyset\}\}$. We will simply write δ without a subscript when this causes no ambiguity. We require that $\delta(\Omega^o, q) = \{\emptyset\}$ if the rank of q is even, and $\delta(\Omega^o, q) = \emptyset$ otherwise[1].

[1] This unusual treatment of Ω^o is a small but important ingredient of our construction. Any other choice looses the correspondance between ranks in $\mathcal{A}$ and fixpoint alternation in the definition of the fixpoint.

A *run of* $\mathcal{A}$ *on* t *from a state* q^0 is a labelling of nodes of t with the states of $\mathcal{A}$ such that: (i) the root is labelled with q^0, (ii) if a node u is labelled q and its k-successors are labelled by $q_1, \dots q_k$, respectively, then $(q_1, \dots, q_k) \in \delta_k(q, t(u))$; recall that $t(u)$ is the letter in the node u.

A run is *accepting* when: (i) for every leaf u of t, if q is the state of the run in u then $\delta_0(q, t(u)) = \{\emptyset\}$, and moreover (ii) for every infinite path of t, the labelling of the path given by the run satisfies the *parity condition*. This means that if we look at the ranks of states assigned to the nodes of the path then the maximal rank appearing infinitely often is even. A tree is *accepted by* $\mathcal{A}$ *from a state* q^0 if there is an accepting run from q^0 on the tree.

It is well known that for every MSOL formula there is a parity automaton recognizing the set of trees that are models of the formula. The converse also holds. Let us also recall that the automata model can be extended to alternating parity automata without increasing the expressive power. Here, for simplicity of the presentation, we will work only with nondeterministic automata but our constructions apply also to alternating automata.

In the context of verification of higher-order properties, automata with *trivial acceptance conditions* have gathered considerable attention [14]. These are obtained by requiring that all states have rank 0. In terms of runs it means that every run of such an automaton on an infinite tree without leaves is accepting. For the reasons that will be apparent in the next subsection one more simplifying condition is imposed in the literature. An automaton is Ω-*blind* if $\delta(q, \Omega) = \{\emptyset\}$ for all states q. So Ω-blind automaton unconditionally accepts divergent computations, while our definition allows to test divergence with the rank of the state.

A parity automaton together with a state *recognize* a language of closed terms of type o:

$$L(\mathcal{A}, q^0) = \{M : M \text{ is closed term of type } o,\ BT(M) \text{ is accepted by } \mathcal{A} \text{ from } q^0\} .$$

2.3 Models with the least fixpoint

A Scott model associates to each type A a finite lattice $\mathcal{D}_A$ in which λY-terms of type A can be interpreted. For a type $B \to C$, this lattice is the set of monotone functions f from $\mathcal{D}_B$ to $\mathcal{D}_C$. The set $\mathcal{D}_{B\to C}$ is ordered pointwise ($f \leq g$ when for every $b \in \mathcal{D}_B$, $f(b) \leq g(b)$) making it a lattice. Constants are interpreted as functions of the right type. Fixpoint operators Y are interpreted as the least fixpoints.

The semantics of a term M of type A in a given valuation υ, denoted $[\![M, \upsilon]\!]$, is an element of $\mathcal{D}_A$. As usual, a valuation is a partial function from variables to elements of the model respecting types: if defined $\upsilon(x^A)$ is an element of $\mathcal{D}_A$. We use $\emptyset$ for the empty valuation. The inductive definition of the semantics is presented in Figure 2. For illustration we have also included a clause for constants. It explains the case when we would like to construct a model from an automaton as stated in the theorem below.

$$\begin{aligned}
[\![x, \upsilon]\!] &= \upsilon(x) \\
[\![a, \upsilon]\!] h_1 \dots h_k &= \{q : \exists_{(q_1,\dots,q_k)\in\delta(q,a)} q_i \in h_i \text{ for all } i\} \\
[\![\lambda x.M, \upsilon]\!] h &= [\![M, \upsilon[h/x]]\!] \\
[\![MN, \upsilon]\!] &= [\![M, \upsilon]\!]([\![N, \upsilon]\!])
\end{aligned}$$

Figure 2 Semantics in a Scott model.

A Scott model can be used to recognize a set of terms. A subset R of $\mathcal{D}_o$ is said to *recognize* the set of closed λY-terms M of type o whose semantics is in R, i.e. $[\![M, \emptyset]\!] \in R$. In this way, finitary Scott models determine a class of languages of λY-terms they recognize. The following theorem characterizes this class.

▶ **Theorem 1** ([22]). *A language of λY-terms is recognized by a boolean combination of Ω-blind automata with trivial acceptance condition iff it is recognized by a Scott model where Y constants are interpreted as the least fixpoint.*

This theorem determines the limits of Scott models with least fixpoints. By duality this also applies to models with greatest fixpoints. So in order to capture more properties we need to be able to construct some other fixpoints.

2.4 The case of terms of order at most 1

The case of Scott models clearly pointed out the challenge in a model construction for all parity automata. In this section we will present the special case of our construction for terms of order at most 1. Such terms have only variables of type o and all their subterms are of type of order at most 1. We will construct a model for an arbitrary parity automaton. The advantage of terms of order at most 1 is that we can describe in a direct way what our semantics expresses. The semantic equations for the general case will be the same as here. We hope that this presentation will give some general intuitions about what properties of Böhm trees the model captures, as well as specific intuitions about the operation $(\cdot)\lfloor_r$ (cf. Definition 3) that deals with parity acceptance conditions at the level of semantics. One can see the construction below as a reformulation of the type system of Kobayashi and Ong [16] in terms of models.

For the rest of the subsection we fix a parity automaton $\mathcal{A} = \langle Q, \Sigma, \delta, rk : Q \to [m] \rangle$.

Let us first consider terms without fixpoints. If M is a closed term of type o then $BT(M)$ is a finite tree with internal nodes labelled by constants of types of order 1 and leaves labelled by constants of type o. It is clear what is an accepting run of automaton $\mathcal{A}$ on $BT(M)$.

Suppose now that M has free variables, that are necessarily of type o. If M is of type o then $BT(M)$ is still a finite tree but it may have nodes labelled by variables. We can thus consider variables as holes where we can put states and ask whether there is a run. The parity condition requires to keep more information. So in addition to states, we keep track of the maximal ranks of states that appear on the paths from the root to the leaves labelled with variables. This idea is formalized in the following definition and illustrated in Figure 3.

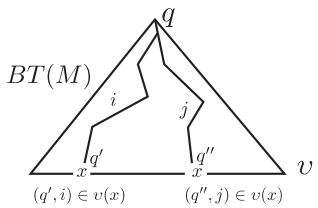

Figure 3 Acceptance from q to υ

▶ **Definition 2.** Let $M : o$ be a term of order at most 1. Let υ be a function assigning to every free variable of M a value from $\mathcal{P}(Q \times [m])$. We say that $\mathcal{A}$ *accepts* $BT(M)$ *from* q *to* υ iff there is a run of $\mathcal{A}$ on $BT(M)$ starting in q, satisfying the conditions of an accepting

run from page 5, and such that for every variable x and leaf of $BT(M)$ labelled by x: if q' is a state of the run in the leaf, and i is the maximal rank of states on the path from the root to the leaf then $(q', i) \in \upsilon(x)$.

We will define a semantics of λ-terms that captures this notion of acceptance. First we define semantic domains for types of order at most 1:

$$\mathcal{D}_o = \mathcal{P}(Q) \qquad \mathcal{R}_o = \mathcal{P}(\{(q,r) : q \in Q \text{ and } rk(q) \leq r \leq m\})$$
$$\mathcal{D}_{o\to\cdots\to o\to o} = \mathcal{R}_o \to \cdots \to \mathcal{R}_o \to \mathcal{D}_o$$

So $\mathcal{R}_o$ is the set of sets of ranked states, with the restriction that the rank should be at least as big as the rank assigned to the state in the automaton. The intended meaning of ranks given by the above definition clearly justifies this restriction. We call the elements of $\mathcal{R}_o$ *residuals*.

Both $\mathcal{D}_o$ and $\mathcal{R}_o$ are ordered by inclusion, and $\mathcal{D}_{o^i\to o}$ is ordered pointwise.

We now introduce the operation $(\cdot)\downarrow_r$ that is taking care of the parity condition at the level of semantics. Even though the definition may at first sight seem technical, Lemma 4 provides some rather clear intuitions about how it works.

▶ **Definition 3.** For $h \in \mathcal{R}_o$, and $r \in [m]$ we put

$$h\downarrow_r = \{(q,i) \in h : r \leq i\} \cup \{(q,j) : (q,r) \in h, rk(q) \leq j \leq r\} .$$

As an example, observe that $h\downarrow_0 = h$.

▶ **Lemma 4.** *For $h \in \mathcal{R}_o$, $q \in Q$, and $r, r_1, r_2 \in [m]$:*

- $(h\downarrow_{r_1})\downarrow_{r_2} = h\downarrow_{\max(r_1,r_2)}$;
- $(q, rk(q)) \in h\downarrow_r$ *iff* $(q, \max(r, rk(q)) \in h$

The above two properties characterize the family of operations $(\cdot)\downarrow_r$. So Definition 3 is imposed on us if we want to have properties listed in the lemma.

The proof of Proposition 5 below, illustrates how we use the two properties from Lemma 4 to capture in a compositional way the acceptance of Böhm trees of Definition 2.

We also use two other operations. The first is a lifting of elements from $\mathcal{D}_o$ to $\mathcal{R}_o$. The second projects an element of $\mathcal{R}_o$ to $\mathcal{D}_o$ by taking a sort of diagonal.

$$f \cdot r = \{(q,r) : q \in f \text{ and } rk(q) \leq r\} \qquad \text{for } f \in \mathcal{D}_o \text{ and } r \in [m]$$
$$h^\partial = \{q : (q, rk(q)) \in h\} .$$

Given a valuation $\upsilon : Vars \to \mathcal{R}_o$ the semantics of a term M of type A is an element $[\![M, \upsilon]\!] \in \mathcal{D}_A$. Its definition is presented in Figure 4. Put next to the semantics in a Scott model from Figure 2, one can clearly see the differences that are due to the presence of ranks. For example, in the variable rule it is necessary to convert the meaning of a variable from $\mathcal{R}_o$ to $\mathcal{D}_o$. Later, in the application rule, it is necessary to lift the meaning of N from $\mathcal{D}_o$ to $\mathcal{R}_o$. The notation $\upsilon\downarrow_r$ means υ where $(.)\downarrow_r$ is applied pointwise.

In our characterization of the semantics we will use *step functions*. For $f_1, \ldots, f_k \in \mathcal{R}_o$ and $q \in \mathcal{D}_o$ we write

$$f_1 \mapsto \ldots \mapsto f_k \mapsto q$$

for the function h of type $\mathcal{R}_o^k \to \mathcal{D}_o$ such that $h(f'_1, \ldots, f'_k) = \{q\}$ if $f'_i \geq f_i$ for all $i = 1, \ldots, k$ and $h(f'_1, \ldots, f'_k) = \emptyset$ otherwise. A step function $f_1 \mapsto \ldots \mapsto f_k \mapsto (q,i)$ for some $(q,i) \in \mathcal{R}_o$ is defined similarly.

$$
\begin{aligned}
[\![x, \upsilon]\!] =& (\upsilon(x))^{\partial} \\
[\![a, \upsilon]\!] h_1 \dots h_k =& \{q : \exists_{(q_1,\dots,q_k)\in\delta(q,a)}\ q_i \in (h_i\restriction_{rk(q)})^{\partial} \text{ for all } i\} \\
[\![\lambda x.M, \upsilon]\!] h =& [\![M, \upsilon[h/x]]\!] \\
[\![MN, \upsilon]\!] =& [\![M, \upsilon]\!] \langle\!\langle N, \upsilon \rangle\!\rangle \quad \text{where } \langle\!\langle N, \upsilon \rangle\!\rangle = \bigvee_{r=0}^{m} \left([\![N, \upsilon\restriction_r]\!] \cdot r \right)
\end{aligned}
$$

Figure 4 Semantics in an extension of the Scott model with ranks

Example: Take a signature with three constants a, b, c of arity $2, 1, 0$, respectively. Consider a parity automaton $\mathcal{A} = \langle \{q_0, q_1\}, \Sigma, \delta, rk : Q \to [1] \rangle$ where the rank of a state is given by its index, and the only pairs for which the value of δ is not $\emptyset$ are $\delta(a, q_0) = Q \times Q$, $\delta(b, q_1) = Q$, and $\delta(c, q_0) = \delta(c, q_1) = \{\emptyset\}$. So from q_0 the automaton recognizes the set of trees with root labelled a and only finitely many b's on every path.

We are going to evaluate the term $a\, x\, (f(b\, x))$ in the model induced by $\mathcal{A}$ and in the valuation υ that maps x to $\{(q_1, 1)\}$ and f to the step function $\{(q_1, 1)\} \mapsto (q_0, 0)$. We get $[\![a\, x(f(b\, x)), \upsilon]\!] = \{q_0\}$ with the following calculation:

$$
\begin{aligned}
[\![x, \upsilon]\!] =& \{q_1\} & \langle\!\langle x, \upsilon \rangle\!\rangle =& \{(q_1, 1)\} \\
[\![b\, x, \upsilon]\!] =& \{q_1\} & \langle\!\langle b\, x, \upsilon \rangle\!\rangle =& \{(q_1, 1)\} \\
[\![f(b\, x), \upsilon]\!] =& \{q_0\} & \langle\!\langle f(b\, x), \upsilon \rangle\!\rangle =& \{(q_0, 0)\} \\
[\![a\, x(f(b\, x)), \upsilon]\!] =& \{q_0\}\ .
\end{aligned}
$$

▶ Proposition 5. $[\![M, \upsilon]\!] \geq f_1 \mapsto \dots \mapsto f_k \mapsto q$ iff for some fresh variables $z_1 \dots z_k$, $\mathcal{A}$ accepts $BT(M z_1 \dots z_k)$ from q to $\upsilon[f_1/z_1 \dots f_k/z_k]$.

Proof. The case of a variable follows by unrolling the definitions. If $BT(M)$ is just the variable, $\mathcal{A}$ accepts $BT(M)$ from q to υ iff $(q, rk(q)) \in \upsilon$. This is because the maximal rank of a state seen from the root of $BT(M)$ to the leaf (which is the same node as the root) is $rk(q)$.

A more interesting case is that of a constant a, say it is of a type $o \to o \to o$. For the left to right implication, suppose $[\![a, \upsilon]\!] \geq f_1 \to f_2 \to q$. We need to show that $a z_1 z_2$ admits a run from q to a valuation $\upsilon[f_1/z_1, f_2/z_2]$. From the definition of the semantics we have $(q_1, q_2) \in \delta(q, a)$ such that $(q_i, rk(q_i)) \in f_i\restriction_{rk(q)}$. By Lemma 4 we get $(q_i, \max(rk(q_i), rk(q))) \in f_i$. So we can take a run on $a z_1 z_2$ assigning q to the root and q_1, q_2 to the leafs labelled z_1, z_2, respectively. Since indeed $\max(rk(q_i), rk(q))$ is the maximal rank seen in the run from the root to z_i this shows that $\mathcal{A}$ accepts $a z_1 z_2$ from q to υ. The other direction is analogous thanks to the equivalence in Lemma 4.

We consider the case of the application. We will only present the left to right direction. Suppose $[\![MN, \upsilon]\!] \geq f_1 \mapsto \dots \mapsto f_k \mapsto q$, and let us look what is the semantics of the application. Since we are considering only terms of order at most 1, N is of type o and $\langle\!\langle N, \upsilon \rangle\!\rangle$ is in $\mathcal{R}_o$. We have $[\![M, \upsilon]\!] \langle\!\langle N, \upsilon \rangle\!\rangle \geq f_1 \mapsto \dots \mapsto f_k \mapsto q$, which is the same as $[\![M, \upsilon]\!] \geq \langle\!\langle N, \upsilon \rangle\!\rangle \mapsto f_1 \mapsto \dots \mapsto f_k \mapsto q$. Now the induction hypothesis tells us that $BT(M z_0 z_1 \dots z_k)$ is accepted by $\mathcal{A}$ from q to $\upsilon[\langle\!\langle N, \upsilon \rangle\!\rangle / z_0, f_1/z_1, \dots f_k/z_k]$. Now let us look what it means that $(q', r') \in \langle\!\langle N, \upsilon \rangle\!\rangle$. By unfolding the definitions we obtain $q' \in [\![N, \upsilon\restriction_{r'}]\!]$. Using the induction hypothesis for N, we have a run of $\mathcal{A}$ on $BT(N)$ from q' to $\upsilon\restriction_{r'}$. From these observations we construct a required run on $BT(MN z_1 \dots z_k)$ from q to υ.

Observe that $BT(MNz_1 \dots z_k)$ is obtained from $BT(Mz_0z_1 \dots z_k)$ by plugging in every leaf labelled z_0 the tree $BT(N)$ (cf. Figure 5). We want to construct on $BT(MNz_1 \dots z_k)$ a run from q to υ. For this we just take a run on $BT(Mz_0z_1 \dots z_k)$ from q to the valuation $\upsilon[\langle\!\langle N, \upsilon \rangle\!\rangle / z_0, f_1/z_1, \dots f_k/z_k]$. Then for every leaf l of $BT(Mz_1 \dots z_k)$ labelled z_0 with q_l the state of the run in l and r_l the maximal rank from the root to l, we prolong the run with the run on $BT(N)$ from q_l to $\upsilon\downarrow_{r_l}$.

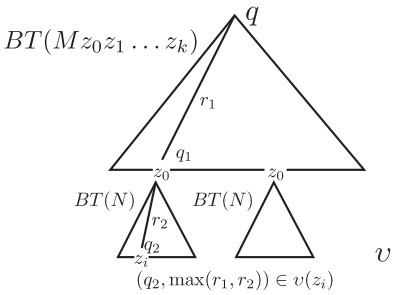

Figure 5 The case of application.

To show that this run is as required we take a leaf l_2 of $BT(MNz_1 \dots z_k)$ labelled by some variable y. We suppose that q_2 is the state assigned by the run to l_2 and that r is the maximal rank of states of the run on the path from the root to l_2. We want to show that $(q_2, r) \in \upsilon(y)$. If l_2 is a leaf of $BT(Mz_0z_1 \dots z_k)$ then this directly follows from the definition of the run. If it is not, then the path to l_2 passes through the leaf l_1 of $BT(Mz_0z_1 \dots z_k)$ labelled by z_0 and then gets to $BT(N)$; cf. Figure 5. Let q_1 be the state labelling l_1, let r_1 be the maximal rank from the root to l_1, and let r_2 be the maximal rank from l_1 to l_2. By looking at the part of the run on $BT(N)$ we get $(q_2, r_2) \in \upsilon\downarrow_{r_1}(y)$. Lemma 4 then gives $(q_2, \max(r_1, r_2)) \in \upsilon(y)$, that is exactly the required property. ◀

The above proof is so simple because the composition of Böhm trees of terms of order at most 1 is easy. We can now try to add a fixpoint to our syntax. We consider terms of the form YM with M of type $o \to o$. The semantics of a term $[\![M, \upsilon]\!]$ is a function from $\mathcal{R}_o$ to $\mathcal{D}_o$. If we want to calculate the semantics of YM then we need to do some manipulation with the function $[\![M, \upsilon]\!]$ as its domain and co-domain are different. The situation becomes clearer when we recall that $\mathcal{R}_o \subseteq \mathcal{D}_o \times [m]$. So $[\![M, \upsilon]\!]$ is essentially a function of m arguments. This is very fortunate as we can expect that the computation of the semantics of YM needs m fixpoints alternating between the least and the greatest fixpoints.

We will give a general formula for calculating the fixpoint in Section 3 when we fully describe our model. Since we have Y in the syntax, this formula itself should denote an element of our model. Here let us show the formula for the case of $m = 1$. This means that we have two ranks 0 and 1. Using $f : \mathcal{R}_o \to \mathcal{D}_o$ to denote the function $[\![M, \upsilon]\!]$ the semantics $[\![YM, \upsilon]\!]$ is given by $F_0 \in \mathcal{D}_o$ defined by

$$F_0 = \nu Z_0.\ f^{\partial}(Z_0 \cdot 0 \cup F_1 \cdot 1)$$
$$F_1 = \mu Z_1 \nu Z_0.\ (f\downarrow_1)^{\partial}(Z_0 \cdot 0 \cup Z_1 \cdot 1)$$

We omit a, not so short, proof of the correctness of this formula. The proof for the general case is presented in the appendix.

3 A model recognizing MSOL properties

We now extend the definitions we have given in the previous section to higher orders. We obtain a model of the λY-calculus that recognizes terms whose Böhm trees are accepted by a given parity automaton. More precisely, for every closed λY-term M of type o we will have:

$$[\![M, \emptyset]\!] = \{q : \mathcal{A} \text{ accepts } BT(M) \text{ from } q\} \ .$$

For the rest of this section we fix a parity automaton $\mathcal{A} = \langle Q, S, \delta, rk : Q \to [m]\rangle$. In particular, m is the maximal rank of a state of $\mathcal{A}$.

We start by generalizing the definition of residuals $\mathcal{R}_o$ to all types. At the same time we will generalize the operation $(\cdot)\lfloor_r$, as well as define a new operation $(\cdot)\Downarrow_q$. For a residual f in $\mathcal{R}_o$, we let $f\Downarrow_q$ be $\{r : (q, r) \in f\}$. Now we define $\mathcal{R}_{A\to B}$ to be the set of monotone functions f that satisfy the following *stratification* property:

$$\forall g \in \mathcal{R}_A.\ \forall q \in Q.\ (f(g))\Downarrow_q = (f(g\lfloor_{rk(q)}))\Downarrow_q \qquad (\texttt{strat})$$

at the same time we define for every $g \in \mathcal{R}_A$:

$$f\Downarrow_q(g) = (f(g))\Downarrow_q, \qquad f\lfloor_r(g) = (f(g))\lfloor_r \ .$$

The elements of $\mathcal{R}_A$ are ordered using the pointwise order. It can be shown that this order makes $\mathcal{R}_A$ a lattice.

For an intuition behind the (strat) property it may be useful to look back at Figure 5. Suppose f is the meaning of M and g is the meaning of N. The formula $f(g)\Downarrow_q$ then means that we are interested in the runs on $BT(MN)$ starting from q. As can be seen from the proof of Proposition 5, in such a run every appearance of $BT(N)$ will be lifted with $\lfloor_r$ operation where r is the maximal rank seen from the root to this appearance. We do not know what this r will be, but it will be at least $rk(q)$, so it is safe to already apply $\lfloor_{rk(q)}$ to g. In other words, for the runs starting in q we should get the same result from $f(g)$ as from $f(g\lfloor_{rk(q)})$. Yet another more formal intuition comes from the application clause. The meaning of $\langle\langle N, \upsilon\rangle\rangle$ as a function of υ satisfies the (strat) property.

As in the previous section, we do not interpret λY-terms in the lattices $\mathcal{R}_A$, but rather in the lattices $\mathcal{D}_A$ that are generalizations at every type of $\mathcal{D}_o$. For this we must define $f\Downarrow_q$ for $f \in \mathcal{D}_A$: we put $f\Downarrow_q = f \cap \{q\}$ for $f \in \mathcal{D}_o$; and $f\Downarrow_q(g) = (f(g))\Downarrow_q$ for $f \in \mathcal{D}_{A\to B}$, and $g \in \mathcal{R}_A$. Using the same notation for the operation $\Downarrow_q$ when it acts on $\mathcal{D}_A$ or $\mathcal{R}_A$ should not confuse the reader as in both cases, it corresponds to focusing on the behaviour of the function on the state q. With this definition we let $\mathcal{D}_{A\to B}$ be the set of monotone functions from $\mathcal{R}_A$ to $\mathcal{D}_B$ that satisfy the same (strat) identity.

Remark: The definitions of $(\cdot)\lfloor_r$ and $(\cdot)\Downarrow_q$ are covariant and they become more intuitive when we consider types written as $A_1 \to \cdots \to A_k \to o$, or in an abbreviated form as $\vec{A} \to o$. In this case, using $\to_{ms}$ for the set of monotone and stratified functions, we have:

$$\mathcal{D}_{\vec{A}\to o} = \mathcal{R}_{A_1} \to_{ms} \cdots \to_{ms} \mathcal{R}_{A_k} \to_{ms} \mathcal{D}_o$$
$$\mathcal{R}_{\vec{A}\to o} = \mathcal{R}_{A_1} \to_{ms} \cdots \to_{ms} \mathcal{R}_{A_k} \to_{ms} \mathcal{R}_o$$
$$g\Downarrow_q(\vec{h}) = (g(\vec{h}))\Downarrow_q \qquad g\lfloor_r(\vec{h}) = (g(\vec{h}))\lfloor_r$$

where $\vec{h}$ is a vector of elements from $\mathcal{R}_{A_1} \times \cdots \times \mathcal{R}_{A_k}$, and the operations $\Downarrow_q$, $\lfloor_r$ are applied only to elements from $\mathcal{D}_o$ or $\mathcal{R}_o$, depending on whether g is from $\mathcal{D}_{\vec{A}\to o}$ or $\mathcal{R}_{\vec{A}\to o}$.

Before we define the semantics, we observe several properties of the domains and the operations we have introduced. First, the generalization of $(\cdot)\downarrow_r$ to higher orders preserves the properties of Lemma 4.

▶ **Lemma 6.** *For every type A, both $\mathcal{D}_A$ and $\mathcal{R}_A$ are finite complete lattices. When A is $A_1 \to \cdots \to A_l \to o$, $g \in \mathcal{R}_A$, $\vec{h} \in \mathcal{R}_{A_1} \times \cdots \times \mathcal{R}_{A_l}$ and $r, r_1, r_2 \in [m]$ then:*

- $(g\downarrow_{r_1})\downarrow_{r_2} = g\downarrow_{\max(r_1,r_2)}$;
- $(q, rk(q)) \in g\downarrow_r(\vec{h})$ *iff* $(q, \max(rk(q), r)) \in g(\vec{h})$.

For every g_1, g_2 in $\mathcal{R}_A$: $(g_1 \vee g_2)\downarrow_r = g_1\downarrow_r \vee g_2\downarrow_r$ and $(g_1 \wedge g_2)\downarrow_r = g_1\downarrow_r \wedge g_2\downarrow_r$.

We now extend to higher-orders the operations $(\cdot)^\partial$ and $(\cdot)\cdot r$ we have introduced in Section 2.4. These extensions use the same covariant pattern as the extensions of $(\cdot)\Downarrow_q$ and $(\cdot)\downarrow_r$; we first define the operations for objects of type o and then extend them to all higher types. For $g_0 \in \mathcal{R}_o$, $f_0 \in \mathcal{D}_o$, $g_1 \in \mathcal{R}_{A\to B}$, $f_1 \in \mathcal{D}_{A\to B}$ we have:

$$g_0^\partial = \{q : (q, rk(q)) \in g_0\} \qquad g_1^\partial(h) = (g_1(h))^\partial$$
$$f_0 \cdot r = \{(q, r) : q \in f_0, rk(q) \leq r\} \qquad (f_1 \cdot r)(h) = (f_1(h)) \cdot r$$

Thus g^∂ converts an element of $\mathcal{R}_A$ to an element of $\mathcal{D}_A$, and $f \cdot r$ does the opposite.

▶ **Lemma 7.** *For every type A, every $f \in \mathcal{D}_A$, $g \in \mathcal{R}_A$, and $r \in [m]$, we have: $f \cdot r \in \mathcal{R}_A$, $g\downarrow_r \in \mathcal{R}_A$, and $g^\partial \in \mathcal{D}_A$.*

The *semantics of a term M* of some type A, under a given valuation v is denoted $[\![M, v]\!]$. It is an element of $\mathcal{D}_A$ provided v is defined for all free variables of M. As in Section 2.4, a valuation is a function assigning to a variable of type B an element of $\mathcal{R}_B$. The semantic clauses are those from Figure 4, so they are the same as for the order 1 case of Section 2.4. It remains to define the fixpoint:

$$[\![Y^A, v]\!]h = \mathrm{fix}(h, 0)$$

where for $l = 0, \ldots, m$ we define

$$\mathrm{fix}(h, l) = \sigma f_l \ldots \mu f_1 . \nu f_0 .\ (h\downarrow_l)^\partial\Big(\bigvee_{i=0}^{l} f_i \cdot i \vee \bigvee_{i=l+1}^{m} \mathrm{fix}(h, i) \cdot i\Big)\ .$$

We use σ to stand for μ or ν depending on whether l is odd or even, respectively.

The structure of this formula may be better visible if we look at $\mathrm{fix}(h, m)$, and assume that m is odd:

$$\mu f_m . \nu f_{m-1} \ldots \mu f_1 . \nu f_0 .\ (h\downarrow_m)^\partial\Big(\bigvee_{i=0}^{m} f_i \cdot i\Big)\ .$$

So we see a rather expected alternation of least and greatest fixpoints, and inside the big brackets we see an operation of composing f_i's to one residual. This operation is of the same shape as in the clause for application. Observe that the expression $(\bigvee_{i=0}^{m} f_i \cdot i)$ considered as a function of $f_0, \ldots, f_m$ is a monotone function from $\mathcal{D}_A^{m+1}$ to $\mathcal{D}_A$. This remark together with Lemma 7 and the fact that $\mathcal{D}_A$ is a complete lattice explains why $\mathrm{fix}(h, l)$ is well-defined, for every l.

We state a couple of lemmas implying that what we have defined is indeed a model.

▶ **Lemma 8.** *For every type A, if f is in $\mathcal{R}_{A\to A}$ then for every $k,l\in[m]$:*
(i) $\mathrm{fix}(f,l)$ *is in* $\mathcal{D}_A$*; and (ii)* $\mathrm{fix}(f\restriction_k,l)=\mathrm{fix}(f,\max(k,l))$.

The next lemma implies that for every term M of type A, the value $[\![M,\upsilon]\!]$ assigned by the semantics is indeed in $\mathcal{D}_A$.
Notation: We write $(\cdot)\restriction_q$ for $(\cdot)\restriction_{rk(q)}$.

▶ **Lemma 9.** *For every term M, every υ, and $\vec{f}$, of appropriate types:*
1. *If $\upsilon\le\upsilon'$ and $\vec{f}\le\vec{g}$ then $[\![M,\upsilon]\!]\vec{f}\le[\![M,\upsilon']\!]\vec{g}$.*
2. *For every $q\in Q$: $q\in[\![M,\upsilon]\!]\vec{f}$ iff $q\in[\![M,\upsilon]\!]\vec{f}\restriction_q$ iff $q\in[\![M,\upsilon\restriction_q]\!]\vec{f}\restriction_q$.*
3. *$[\![M,\upsilon]\!]$ and $\langle\!\langle M,\upsilon\rangle\!\rangle$ satisfy the* (`strat`) *property.*
4. *$\langle\!\langle M,\upsilon\restriction_q\rangle\!\rangle=\langle\!\langle M,\upsilon\rangle\!\rangle\restriction_q$.*

The above lemmas allow us to show that the interpretation of terms is invariant under $=_{\beta\delta}$, or, put differently, that we have constructed a model of λY-calculus.

▶ Proposition 10. For every M, N and υ, if $M=_{\beta\delta}N$, then $[\![M,\upsilon]\!]=[\![N,\upsilon]\!]$ and $\langle\!\langle M,\upsilon\rangle\!\rangle=\langle\!\langle N,\upsilon\rangle\!\rangle$.

It now remains to explain how this model is related to the acceptance of the Böhm trees of λY-terms by $\mathcal{A}$. This explanation is given by the following theorem which is the main result of the paper. Recall that we denote the empty valuation by $\emptyset$.

▶ **Theorem 11** (Correctness). *For a given parity automaton $\mathcal{A}$, the semantics defined above is such that for every closed term M of type o and every state q of $\mathcal{A}$:*

$q\in[\![M,\emptyset]\!]$ *iff $\mathcal{A}$ accepts $BT(M)$ from state q.*

Example: Continuing the example from page 8 we will calculate the value of the term $M^{o\to o}=Y(\lambda fx.a\,x\,(f(b\,x)))$. This term is a simplified version of `map` function from the Introduction, in the sense that it has a Böhm tree of a similar shape. In order to show that every path of $BT(Mc)$ contains only finitely many b's we show $q_0\in[\![Mc,\emptyset]\!]$. In the first part of the example we have established $[\![a\,x\,(f(b\,x)),\upsilon]\!]=\{q_0\}$ where υ that maps x to $\{(q_1,1)\}$ and f to the step function $\{(q_1,1)\}\mapsto(q_0,0)$. This implies that $[\![\lambda fx.a\,x(f(b\,x))]\!]\ge g$ where $g=(\{(q_1,1)\}\mapsto\{(q_0,0)\})\mapsto\{(q_1,1)\}\mapsto q_0$. We now compute $\mathrm{fix}(g,0)$. We observe that $g(\top\cdot 0\vee\bot\cdot 1)=\{(q_1,1)\}\mapsto q_0$ and, $g(h\cdot 0\vee\bot\cdot 1)=h$, for $h=\{(q_1,1)\}\mapsto q_0$. Therefore $\nu g_0.g(g_0\cdot 0\vee\bot\cdot 1)=h$. Now, $g(h\cdot 0\vee h\cdot 1)=h$ which implies that $\mu g_1.\nu g_0.g(g_0\cdot 0\vee g_1\cdot 1)=h$. With this we have showed $[\![M,\emptyset]\!]\ge h$ which finally gives us $q_0\in[\![Mc]\!]$.

The proof of Theorem 11 is presented in the appendix.

4 Applications

The model construction we have presented allows us to derive a number of results on verification of higher-order schemes and the λY-calculus. Since the constructed model is finite, it implies the decidability of the model-checking problem [19]. More importantly, it implies the transfer theorem [21]. Actually this theorem is proved in op. cit. also for infinite terms. This cannot be done solely with the techniques in the present paper. The transfer theorem gives an effective reduction of the MSOL theory $BT(M)$ to the MSOL theory of the tree representation of M. The strength of the theorem lies in the fact that the reduction is uniform for all terms over a fixed set of variables and types.

A term can be represented as a tree with back edges: the nodes of the tree are labelled with the application symbol, the lambda abstraction, a variable, or a constant. The back

edges go from occurrences of variables to their binding lambdas. This representation makes it rather clear what it means for a term to be a model of an MSOL formula [21]. We will use $Terms(\Sigma, \mathcal{T}, X)$ for the set of terms over a signature Σ, such that all the their (free or bound) variables are from $\mathcal{X}$, and all their subterms have types in $\mathcal{T}$.

▶ **Theorem 12** ([21]). *Let Σ be a finite tree signature, $\mathcal{X}$ a finite set of typed variables, and $\mathcal{T}$ a finite set of types. For every MSOL formula φ one can effectively construct an MSOL formula $\widehat{\varphi}$ such that for every λY-term $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$ of type o:*

$$BT(M) \vDash \varphi \qquad \textit{iff} \qquad M \vDash \widehat{\varphi}.$$

Proof. Let $\mathcal{A}$ be the automaton equivalent to φ. Consider the model $\mathcal{D}^{\mathcal{A}}$ given by Theorem 11. The model $\mathcal{D}^{\mathcal{A}}$ is finite in every type. So the set of possible semantical values of terms from $Term(\Sigma, \mathcal{T}, \mathcal{X})$ is finite.

There is a correspondence between subterms of the term and the nodes of the tree representation of the term. So the labelling assigning to a node of the tree representation the meaning of the subterm it represents is a colouring of the tree with colours from a finite set. Let us call it the *semantic colouring*. The next observation is that if we are given any colouring of the tree representation of a term with elements of the model then we can check if it is the semantic colouring by verifying some local constraints implied by the definition of the model. For example, the local constraints say that the meaning assigned to the node labelled by the application symbol is indeed the result of the application of the meaning assigned to the first child applied to the meaning assigned to the second child. This can be checked by a looking in a finite table. Now the desired MSOL formula can guess such a colouring of the tree representation of a term, verify that it satisfies the local constraints, and that the initial state of the automaton $\mathcal{A}$ belongs to the colour of the root node. ◀

This theorem implies the global model checking property [2]. Other applications of the theorem are outlined in [21]. In particular, a model clearly explains how to solve the synthesis problem from higher-order modules [21]. The synthesized program is composed from modules using application. Since the set of modules is fixed and finite, we can evaluate the meaning of such a composition using a finite automaton. Thus the synthesis problem is reduced to the emptiness problem for finite automata on finite trees.

As described in [22], a model can be used to design program transformations. A general principle of such a transformation is that during evaluation the program "knows" what is its meaning in the model. Such a program, or in our case a term of λY-calculus, is called *reflective* [1]. This intuitive statement requires some explanation. What we mean is that when evaluating a term M we reach a head normal form, say bN_1N_2. Then b is a non-interpreted symbol that is output as the root of the tree $BT(M)$, and the evaluation process splits to evaluation of N_1 and N_2. While at the beginning we can simply calculate the semantics $[\![M]\!]$ in the model, it is the reflective program itself that needs to calculate $[\![N_1]\!]$ and $[\![N_2]\!]$. As the model is finite, Church encoding permits to work with model's elements. Interestingly, this general method of translating a term into a reflective term follows a simple inductive pattern. We refer to [22] for more details.

5 Conclusions

We have extended Scott models with ranks, and have shown that this extension recognizes all MSOL properties of λY-terms. The meaning of the fixpoint operator is an alternation of

the least and the greatest fixpoints reminiscent to the fixpoint characterization of winning positions in a parity game. This is somehow expected since acceptance for parity automata is expressed in terms of existence of a strategy in a parity game.

The model construction reduces the higher-order verification problem to the evaluation problem. Surprisingly, even the problem of evaluating terms without fixpoints in a Scott model is not that well studied (cf. [24]). We believe that the evaluation problem can be an unifying algorithmic problem for many kinds of program analyses whose theoretical complexity is "sufficiently high" to justify a semantic approach. Verification of MSOL properties considered in this paper is one such case. The model we construct is essentially of the same size as the Scott model so the evaluation approach should be essentially as efficient as approaches based on intersection types refining simple types. Indeed, every step function in the model can be represented by such a type.

Model formulation opens a possibility to use abstract interpretation methods. For example, instead of λY-calculus one can consider PCF extended with uninterpreted constants. So a PCF term will also generate an infinite tree constructed from added constants. In order to evaluate such PCF terms we would need to work with infinite models with fixpoints of the form we have here. Under some conditions these models could recognize MSOL properties of Böhm trees generated by such terms. Then abstract interpretation will be given by a suitable homomorphisms from infinite to finite models.

We hope that our result is a step towards understanding infinitary properties in the usual frameworks of semantics, and with this to extend semantic methods to reactive programs and their behaviors. We have tried here to make the presentation as concrete as possible. It is evident though that a more abstract description bringing out the structure of the model should be pursued. A more ambitious goal is to find an abstract description of models recognizing MSOL properties. Let us mention that the expressive power of Scott models with arbitrary interpretations of fixpoints is unknown.

References

1 C. Broadbent, A. Carayol, L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129, 2010.

2 C. Broadbent and C.-H. L. Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, volume 5504 of *LNCS*, pages 107–121, 2009.

3 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24. ACM, 2013.

4 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

5 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS'91*, pages 368–377, 1991.

6 J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of computer and system sciences*, 15:328–353, 1977.

7 J. Engelfriet and E. M. Schnidt. IO and OI. II. *Journal of computer and system sciences*, 16:67–99, 1978.

8 C. Grellois and P.A. Mellies. An infinitary model of linear logic. In *FoSSACS 2015*, to appear.

9 A. Haddad. Model checking and functional program transformations. In *FSTTCS*, volume 24 of *LIPIcs*, pages 115–126, 2013.

10 M. Hofmann and W. Chen. Abstract interpretation from büchi automata. In *LICS-CSL*, pages 51:1–51:10, 2014.

11 Yu I Ianov. The logical schemes of algorithms. *Problems of cybernetics*, 1:82–140, 1960.

12 A Jeffrey. Functional reactive types. In *CSL-LICS*, pages 54:1–54:10, 2014.

13 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303, pages 205–222, 2002.

14 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428, 2009.

15 N. Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20–89, 2013.

16 N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.

17 P.A. Mellies. private communication, June 2014.

18 M. Naik and J. Palsberg. A type system equivalent to a model checker. *ACM Trans. Program. Lang. Syst.*, 30(5), 2008.

19 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

20 S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, pages 61–72. ACM, 2014.

21 S. Salvati and I. Walukiewicz. Evaluation is MSOL-compatible. In *FSTTCS*, volume 24 of *LIPIcs*, pages 103–114, 2013.

22 S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, volume 7941 of *LNCS*, pages 189–204, 2013.

23 S. Salvati and I. Walukiewicz. Typing weak msol properties. In *FoSSACS 2015*, to appear. https://hal.archives-ouvertes.fr/hal-01061202v2.

24 K. Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl, 2012.

25 T. Tsukada and C.-H. L. Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *LICS-CSL*, pages 78:1–78:10, 2014.

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

A Omitted proofs from Section 3

Lemma 6 For every type A, both $\mathcal{D}_A$ and $\mathcal{R}_A$ are finite complete lattices. When A is $A_1 \to \cdots \to A_l \to o$, $g \in \mathcal{R}_A$, $\vec{h} \in \mathcal{R}_{A_1} \times \cdots \times \mathcal{R}_{A_l}$ and $r, r_1, r_2 \in [m]$ then:

- $(g\lfloor_{r_1})\lfloor_{r_2} = g\lfloor_{\max(r_1,r_2)}$;
- $(q, rk(q)) \in g\lfloor_r(\vec{h})$ iff $(q, \max(rk(q), r)) \in g(\vec{h})$.

For every g_1, g_2 in $\mathcal{R}_A$:
$(g_1 \vee g_2)\lfloor_k = g_1\lfloor_k \vee g_2\lfloor_k$ and $(g_1 \wedge g_2)\lfloor_k = g_1\lfloor_k \wedge g_2\lfloor_k$.

Proof. Proving that $\mathcal{D}_A$ and $\mathcal{R}_A$ are finite complete lattices is a simple induction on the structure of A.

Since by definition $g\lfloor_r(\vec{h}) = (g(\vec{h}))\lfloor_r$, the argument for higher types is the same as for the base type. So we will consider only g of type o.

If (q, r) is in $(g\lfloor_{r_1})\lfloor_{r_2}$, then either $r \geq r_2$ and (q, r) is in $g\lfloor_{r_1}$, or $r < r_2$ and (q, r_2) is in $g\lfloor_{r_1}$. Each case can then be split in two.

In the first case, (q, r) is in $g\lfloor_{r_1}$ either since $r \geq r_1$ and (q, r) is in g, or since $r < r_1$ and (q, r_1) is in g. If $r \geq r_1$ and (q, r) is in g, then, because $r \geq r_2$, we have $r \geq \max(r_1, r_2)$ and thus, (q, r) is in $g\lfloor_{\max(r_1,r_2)}$. If $r < r_1$, then because $r \geq r_2$, we have $\max(r_1, r_2) = r_1$, then because (q, r) is in $g\lfloor_{r_1}$, we also have that (q, r) in $g\lfloor_{\max(r_1,r_2)}$.

In the second case, (q, r_2) is in $g\lfloor_{r_1}$ either since $r_2 \geq r_1$ and (q, r_2) is in g or since $r_2 < r_1$ and (q, r_1) is in g. If $r_2 \geq r_1$, then $r_2 = \max(r_1, r_2)$, and since (q, r_2) is in is in g, because $r < r_2$, we obtain that (q, r) is in $g\lfloor_{\max(r_1,r_2)}$. If $r_2 < r_1$, then $r_1 = \max(r_1, r_2)$ and as, (q, r_1) is in g, we obtain that (q, r) is in $g\lfloor_{\max(r_1,r_2)}$.

We have thus showed that $(g\lfloor_{r_1})\lfloor_{r_2} \subseteq g\lfloor_{\max(r_1,r_2)}$. We now turn to the converse inclusion. If (q, r) is in $g\lfloor_{\max(r_1,r_2)}$, then it is either because $r \geq \max(r_1, r_2)$ and (q, r) is in g or because $r < \max(r_1, r_2)$ and $(q, \max(r_1, r_2))$ is in g. In both cases, the definitions immediately lead to the fact that (q, r) is also in $(g\lfloor_{r_1})\lfloor_{r_2}$. In the first case, since $r \geq \max(r_1, r_2)$ and (q, r) is in g, we have that (q, r) is in $g\lfloor_{r_1}$ which finally entails that (q, r) is in $(g\lfloor_{r_1})\lfloor_{r_2}$. In the second case, we have that $(q, \max(r_1, r_2))$ is in g which implies that for every $r' \leq \max(r_1, r_2)$, (q, r') is in $(g\lfloor_{r_1})\lfloor_{r_2}$ and, thus, (q, r) is in $(g\lfloor_{r_1})\lfloor_{r_2}$.

The second and third statements follow the similar reasoning but are easier. ◀

As a side remark we show that the operations $(\cdot)\lfloor_r$ is characterised by the two properties from the previous lemma.

▶ **Lemma 13.** *Suppose that we have an operation $lift_k : [m] \times \mathcal{R}_o \to \mathcal{R}_o$ satisfying the conditions:*

- $lift(r_1, lift(r_2, g)) = lift(\max(r_1, r_2), g)$;
- $(q, rk(q)) \in lift(r, g)$ *iff* $(q, \max(rk(q), r)) \in g$.

then $lift(k, g) = g\lfloor_k$ for every $k \in [m]$

Proof. Take some $r \in [m]$. Suppose $(q, i) \in lift(r, g)$. By the second condition we know that this is equivalent to $(q, rk(q)) \in lift(i, lift(r, g)) = lift(\max(i, r), g)$; where the equality is by the first condition. We have two cases. If $i \geq r$ then $(q, rk(q)) \in lift(i, g)$, By the second condition this is equivalent to $(q, i) \in g$. If $i < r$ then $(q, rk(q)) \in lift(r, g)$. So this time it is equivalent to $(q, r) \in g$. So in both cases this is equivalent to $(q, i) \in g\lfloor_r$. ◀

Lemma 7 For every type A, every $f \in \mathcal{D}_A$, $g \in \mathcal{R}_A$, and $r \in [m]$, we have: $f \cdot r \in \mathcal{R}_A$, $g\lfloor_r \in \mathcal{R}_A$, and $g^\partial \in \mathcal{D}_A$.

Proof. When $A = o$ is the base type, the statements are immediate. We verify that $f \cdot r$ satisfies the (`strat`) condition when A is a higher-order type. For the induction step it is convenient to consider types in the form $\vec{A} \to o$. Take a vector of elements $\vec{h}$ of types $\vec{A}$ and a state q. We have $(f \cdot r)(\vec{h}) = (f(\vec{h})) \cdot r$, and similarly $(f \cdot r)(\vec{h}\downarrow_q) = (f(\vec{h}\downarrow_q)) \cdot r$. So we need to show that $(f(\vec{h}) \cdot r)\Downarrow_q = ((f(\vec{h}\downarrow_q)) \cdot r)\Downarrow_q$. To show the inclusion from left to right, take $i \in (f(\vec{h}) \cdot r)\Downarrow_q$. We must have $(q, i) \in f(\vec{h}) \cdot r$ so $i = r$, $rk(q) \geq r$ and $q \in f(\vec{h})$. The (`strat`) property gives $q \in f(\vec{h}\downarrow_q)$ implying that r belongs to $((f(\vec{h}\downarrow_q)) \cdot r)\Downarrow_q$. The other direction is similar. By the same argument if $i \in ((f(\vec{h}\downarrow_q)) \cdot r)\Downarrow_q$ then $i = r$, $rk(q) \geq r$ and $q \in f(\vec{h}\downarrow_q)$. Stratification gives $q \in f(\vec{h})$ and we are done.

The reasoning in the case of $\downarrow_r$ operation is similar. We need to show stratification, which means that for every $\vec{h}$ of appropriate types and q we need to show $(g(\vec{h}))\downarrow_r\Downarrow_q = g(\vec{h}\downarrow_q)\downarrow_r\Downarrow_q$. For the left to right inclusion we take $i \in (g(\vec{h}))\downarrow_r\Downarrow_q$. This gives us $(q, i) \in g(\vec{h})\downarrow_r$. If $i \geq r$ then $(q, i) \in g(\vec{h})$ and by (`strat`) also $(q, i) \in g(\vec{h}\downarrow_q)$ so we are done. If $i < r$ then $(q, r) \in g(\vec{h})$ and the same argument works. The right to left inclusion follows with the same lines.

The statement $g^\partial \in \mathcal{D}_A$ follows directly from the definitions. ◀

Lemma 8 For every type A, if f is in $\mathcal{R}_{A\to A}$ then for every $k, l \in [m]$: (i) $\mathrm{fix}(f, l)$ is in $\mathcal{D}_A$; (ii) $\mathrm{fix}(f\downarrow_k, l) = \mathrm{fix}(f, \max(k, l))$.

Proof. We show the first statement of the lemma by induction on $m - l$. In the case where $l = m$, as $\mathcal{D}_A$ is finite, we can rewrite $\mathrm{fix}(f, m)$ as a finite formula with meets and joins of elements of $\mathcal{D}_A$. Using inductively Lemma 6 on this formula proves that indeed $\mathrm{fix}(f, m)$ is in $\mathcal{D}_A$. For the inductive case, the method is the same, but we need the induction hypothesis, Lemma 6 and Lemma 7 to prove that $\bigvee_{i=l+1}^{m} \mathrm{fix}(f, i) \cdot i$ is in $\mathcal{R}_A$.

For the second statement, we first remark that Lemma 6 implies $\mathrm{fix}(f\downarrow_k, l) = \mathrm{fix}(f\downarrow_{\max(k,l)}, l)$. We then prove $\mathrm{fix}(f\downarrow_{\max(k,l)}, l) = \mathrm{fix}(f, \max(k, l))$. We start by showing $\mathrm{fix}(f\downarrow_{k+1}, k) = \mathrm{fix}(f, k + 1)$:

$$
\begin{aligned}
&\mathrm{fix}(f, k+1) = \\
&= \sigma g_{k+1} \ldots \mu g_1 . \nu g_0 . \\
&\qquad (f\downarrow_{k+1})^\partial \left(\bigvee_{i=0}^{k+1} g_i \cdot i \vee \bigvee_{i=k+2}^{m} \mathrm{fix}(f, i) \cdot i \right) \\
&= \sigma' g_k \ldots \mu g_1 . \nu g_0 . \\
&\qquad (f\downarrow_{k+1})^\partial \left(\bigvee_{i=0}^{k} g_i \cdot i \vee \bigvee_{i=k+1}^{m} \mathrm{fix}(f, i) \cdot i \right) \\
&= \sigma' g_k \ldots \mu g_1 . \nu g_0 . \\
&\qquad ((f\downarrow_{k+1})\downarrow_k)^\partial \left(\bigvee_{i=0}^{k} g_i \cdot i \vee \bigvee_{i=k+1}^{m} \mathrm{fix}(f, i) \cdot i \right) \\
&= \mathrm{fix}(f\downarrow_{k+1}, k)
\end{aligned}
$$

In the above the passage from the third to the fourth line follows from Lemma 6. Now, with this identity and by induction on i, we obtain $\mathrm{fix}(f, k + i) = \mathrm{fix}(f\downarrow_{k+i}, k)$ which entails $\mathrm{fix}(f\downarrow_{\max(k,l)}, l) = \mathrm{fix}(f, \max(k, l))$. ◀

▶ **Lemma 14.** *For every M, υ, and $q \in Q$, $\langle\!\langle N, \upsilon\lfloor_q \rangle\!\rangle = \langle\!\langle N, \upsilon \rangle\!\rangle\lfloor_q$.*

Proof.

$$\begin{aligned}
\langle\!\langle N, \upsilon \rangle\!\rangle\lfloor_q =& (\bigvee_{r=0}^{m} [\![N, \upsilon\lfloor_r]\!] \cdot r)\lfloor_q \\
=& (\bigvee_{r>rk(q)}^{m} [\![N, \upsilon\lfloor_r]\!] \cdot r) \vee (\bigvee_{r\leq rk(q)} [\![N, \upsilon\lfloor_q]\!] \cdot r) \\
=& (\bigvee_{r>rk(q)}^{m} [\![N, \upsilon\lfloor_q\lfloor_r]\!] \cdot r) \vee (\bigvee_{r\leq rk(q)} [\![N, \upsilon\lfloor_q\lfloor_r]\!] \cdot r) \\
=& (\bigvee_{r=0}^{m} [\![N, (\upsilon\lfloor_q)\lfloor_r]\!] \cdot r) = \langle\!\langle N, \upsilon\lfloor_q \rangle\!\rangle
\end{aligned}$$

The equality between the first and the second line is a consequence of Lemma 6. ◀

Lemma 9 For every term M, every υ, and $\vec{f}$, of appropriate types:

1. If $\upsilon \leq \upsilon'$ and $\vec{f} \leq \vec{g}$ then $[\![M, \upsilon]\!]\vec{f} \leq [\![M, \upsilon']\!]\vec{g}$.
2. For every $q \in Q$:
   $q \in [\![M, \upsilon]\!]\vec{f}$ iff $q \in [\![M, \upsilon]\!]\vec{f}\lfloor_q$ iff $q \in [\![M, \upsilon\lfloor_q]\!]\vec{f}\lfloor_q$.
3. $[\![M, \upsilon]\!]$ and $\langle\!\langle M, \upsilon \rangle\!\rangle$ satisfy the (`strat`) property.
4. $\langle\!\langle M, \upsilon\lfloor_q \rangle\!\rangle = \langle\!\langle M, \upsilon \rangle\!\rangle\lfloor_q$.

Proof. We show the statements by an induction on the structure of M. The first statement is immediate.

The third statement is implied by the second, since $q \in [\![M, \upsilon]\!](\vec{h})$ iff $q \in [\![M, \upsilon]\!](\vec{h}\lfloor_q)$. Similarly $(q, r) \in \langle\!\langle M, \upsilon \rangle\!\rangle(\vec{h})$ iff $q \in [\![M, \upsilon\lfloor_r]\!](\vec{h})$ iff $q \in [\![M, \upsilon\lfloor_r]\!](\vec{h}\lfloor_q)$ iff $(q, r) \in \langle\!\langle M, \upsilon \rangle\!\rangle(\vec{h}\lfloor_q)$.

It remains to prove the second statement. The only interesting cases are the ones where the term is an application or is equal to Y. We start with the application case. By definition of the semantics

$$q \in [\![MN, \upsilon]\!]\vec{f} \quad \text{iff} \quad q \in [\![M, \upsilon]\!]\langle\!\langle N, \upsilon \rangle\!\rangle\vec{f}\,.$$

The latter, by the induction hypothesis for M, is equivalent to

$$q \in [\![M, \upsilon\lfloor_q]\!]\langle\!\langle N, \upsilon \rangle\!\rangle\lfloor_q\vec{f}\lfloor_q = [\![M, \upsilon\lfloor_q]\!]\langle\!\langle N, \upsilon\lfloor_q \rangle\!\rangle\vec{f}\lfloor_q = [\![MN, \upsilon\lfloor_q]\!]\vec{f}\lfloor_q$$

where the first equality uses Lemma 14. This shows $q \in [\![M, \upsilon]\!]\vec{f}$ iff $q \in [\![M, \upsilon\lfloor_q]\!]\vec{f}\lfloor_q$. Using the fact that $(f\lfloor_q)\lfloor_q = f\lfloor_q$ we then obtain $q \in [\![M, \upsilon\lfloor_q]\!]\vec{f}\lfloor_q$ iff $q \in [\![M, \upsilon\lfloor_q]\!](\vec{f}\lfloor_q)\lfloor_q$ iff $q \in [\![M, \upsilon]\!]\vec{f}\lfloor_q$.

Let us now turn to the case where the term is equal to Y. Because $[\![Y, \upsilon]\!] = [\![Y, \upsilon\lfloor_q]\!]$, it is enough to show the equivalence: $q \in [\![Y, \upsilon]\!]f\vec{h}$ iff $q \in [\![Y, \upsilon]\!]f\lfloor_q\vec{h}\lfloor_q$. By definition $[\![Y, \upsilon]\!]f\vec{h} = \mathrm{fix}(f, 0)\vec{h} = f^{\partial}(\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\vec{h}$. As f is in $\mathcal{R}_{A\to A}$, Lemma 7 tells us that f^{∂} is in $\mathcal{D}_{A\to A}$, therefore, by (`strat`), $q \in f^{\partial}(\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\vec{h}$ iff $q \in f^{\partial}((\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\lfloor_q)\vec{h}\lfloor_q$ iff $(q, rk(q)) \in f((\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\lfloor_q)\vec{h}\lfloor_q$ iff $(q, rk(q)) \in f\lfloor_q((\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\lfloor_q)\vec{h}\lfloor_q$ iff $q \in (f\lfloor_q)^{\partial}((\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i)\lfloor_q)\vec{h}\lfloor_q$. But,

$$\left(\bigvee_{i=0}^{m} \mathrm{fix}(f, i) \cdot i\right)\lfloor_q = \bigvee_{i=0}^{rk(q)} \mathrm{fix}(f, rk(q)) \cdot i \vee \bigvee_{i=rk(q)+1}^{m} \mathrm{fix}(f, i) \cdot i$$

So we have

$$\begin{aligned}
&(f\lfloor_q)^\partial \left(\left(\bigvee_{i=0}^{m} \mathrm{fix}(f,i)\cdot i\right)\lfloor_q\right)(\vec{h}\lfloor_q) = \\
&= (f\lfloor_q)^\partial \left(\bigvee_{i=0}^{rk(q)} \mathrm{fix}(f,rk(q))\cdot i \vee \bigvee_{i=rk(q)+1}^{m} \mathrm{fix}(f,i)\cdot i\right)\vec{h}\lfloor_q \\
&= \mathrm{fix}(f,rk(q))\vec{h}\lfloor_q \\
&= \mathrm{fix}(f\lfloor_q,0)\vec{h}\lfloor_q \\
&= [\![Y,v]\!]f\lfloor_q h\lfloor_q
\end{aligned}$$

Lemma 8 justifies the equality between the second and the third lines. As desired, we have proved that q is in $[\![Y,v]\!]f\vec{h}$ iff it is in $[\![Y,v]\!]f\lfloor_q h\lfloor_q$. ◀

Next we show that the model is invariant under β-reduction.

▶ **Lemma 15.** *For every M and v, $[\![(\lambda x.M)N,v]\!] = [\![M[N/x],v]\!]$.*

Proof. A simple induction on the structure of M yields $[\![M[N/x],v]\!] = [\![M,v[\langle\!\langle N,v\rangle\!\rangle/x]]\!]$ and $\langle\!\langle M[N/x],v\rangle\!\rangle = \langle\!\langle M,v[\langle\!\langle N,v\rangle\!\rangle/x]\rangle\!\rangle$. Once this identity is established, we can obtain $[\![(\lambda x.M)N,v]\!] = [\![\lambda x.M,v]\!]\langle\!\langle N,v\rangle\!\rangle = [\![M,v[\langle\!\langle N,v\rangle\!\rangle/x]]\!] = [\![M[N/x],v]\!]$. ◀

The case of δ reduction is more complicated. We will need two observations.

▶ **Lemma 16.** *For every M and v, $\langle\!\langle M,v\rangle\!\rangle^\partial = [\![M,v]\!]$.*

Proof. We have: $(q,rk(q))$ is in $\langle\!\langle M,v\rangle\!\rangle\vec{h}$ iff q is in $[\![M,v\lfloor_q]\!]\vec{h}$ which, thanks to the Lemma 9, is equivalent to q being in $[\![M,v\lfloor_q]\!]\vec{h}\lfloor q$ and then to q being in $[\![M,v]\!]\vec{h}$. ◀

▶ **Corollary 17.** *For every M and v, $[\![YM,v]\!] = [\![M(YM),v]\!]$.*

Proof. This is obtained with the following sequence of equalities:

$$\begin{aligned}
[\![M(YM),v]\!] &= [\![M,v]\!](\langle\!\langle YM,v\rangle\!\rangle) \\
&= [\![M,v]\!]\left(\bigvee_{i=0}^{m} [\![YM,v\lfloor_i]\!]\cdot i\right) \\
&= [\![M,v]\!]\left(\bigvee_{i=0}^{m} \mathrm{fix}(\langle\!\langle M,v\lfloor_i\rangle\!\rangle,0)\cdot i\right) \\
&= [\![M,v]\!]\left(\bigvee_{i=0}^{m} \mathrm{fix}(\langle\!\langle M,v\rangle\!\rangle\lfloor_i,0)\cdot i\right) && \text{by Lemma 14} \\
&= [\![M,v]\!]\left(\bigvee_{i=0}^{m} \mathrm{fix}(\langle\!\langle M,v\rangle\!\rangle,i)\cdot i\right) && \text{by Lemma 8} \\
&= \langle\!\langle M,v\rangle\!\rangle^\partial\left(\bigvee_{i=0}^{m} \mathrm{fix}(\langle\!\langle M,v\rangle\!\rangle,i)\cdot i\right) && \text{by Corollary 16} \\
&= \mathrm{fix}(\langle\!\langle M,v\rangle\!\rangle,0) \\
&= [\![Y,v]\!](\langle\!\langle M,v\rangle\!\rangle) \\
&= [\![YM,v]\!]
\end{aligned}$$

◀

Proposition 10 For every M, N and υ, if $M =_{\beta\delta} N$, then $[\![M, \upsilon]\!] = [\![N, \upsilon]\!]$ and $\langle\!\langle M, \upsilon \rangle\!\rangle = \langle\!\langle N, \upsilon \rangle\!\rangle$

Proof. Since the λY-calculus enjoys the Church-Rosser property, it suffices to prove the statement when $M \xrightarrow{*}_{\beta\delta} N$. This statement is established by iterating the property: $M \to_{\beta\delta} N$ implies $[\![M, \upsilon]\!] = [\![N, \upsilon]\!]$. The statment is proved by induction on the structure of M. ◀

B Proof of Theorem 11

In Section 3 we have presented a construction of a model from a given automaton. Here we will outline the proof of Theorem 11 characterizing the expressive power of this model. The missing proofs are presented in the next section.

For the rest of this section we fix a parity automaton $\mathcal{A} = \langle Q, \Sigma, \delta, rk : Q \to [m] \rangle$.

Theorem 11 talks only about terms of type o. In order to prove the theorem we need to generalize the statement to terms of arbitrary types. For this we will introduce a special game $\mathcal{G}(\mathcal{A})$ (Definition 18) and show the characterization of the semantics in terms of winning positions in this game (Theorem 19). After this step, we relate winning positions in $\mathcal{G}(\mathcal{A})$ and the acceptance of Böhm trees by $\mathcal{A}$. For this we establish a refinement of an earlier result [21] that characterizes acceptance by $\mathcal{A}$ in terms of some more liberal game $\mathcal{G}^{st}(\mathcal{A})$ (Theorem 23). Finally, we show the equivalence of $\mathcal{G}(\mathcal{A})$ and $\mathcal{G}^{st}(\mathcal{A})$ for positions that interest us (Proposition 24).

B.1 Game $\mathcal{G}(\mathcal{A})$

Our first goal is to generalize the statement from Theorem 11 to terms of all types. For this we introduce a game $\mathcal{G}(A)$ and characterize the semantics with regard to winning positions in this game.

It is easier to present games using a different syntax for fixpoints. We write $Y\mathbf{x}.M$ instead of $Y(\lambda x.M)$. So we consider Y as a binder and not as a constant. Since a term YM is equivalent to $Y\mathbf{x}.M\mathbf{x}$ this is not a restriction. In $Y\mathbf{x}.M$ we use special bold variables that are disjoint from variables bound by λ. Moreover we assume that the variable $\mathbf{x}$ determines M uniquely, that is we have a function $term(\mathbf{x})$ from fixpoint variables to terms such that $term(\mathbf{x})$ is M.

Recall from page 10 that for a type $A = B_1 \to \cdots \to B_k \to o$, the set $\mathcal{D}_A$ is the set of monotone functions in $\mathcal{R}_{B_1} \to_{ms} \cdots \to_{ms} \mathcal{R}_{B_k} \to_{ms} \mathcal{D}_o$ satisfying the (`strat`) property. We use expressions of the form $h_1 \mapsto \cdots \mapsto h_k \mapsto q$ to denote step functions as explained on page 7. Not all such expressions determine functions satisfying (`strat`) property. It can observed though that all of the form $h_1\restriction_q \mapsto \cdots \mapsto h_k\restriction_q \mapsto q$ do.

▶ **Definition 18.** The game $\mathcal{G}(\mathcal{A})$ has positions of the form $(N^A, \upsilon) \geq f$ where N^A is a term of type A, $f \in \mathcal{D}_A$ is a step function, and υ is a valuation assigning to a variable x of a type C a step function in $\mathcal{D}_C$. The game has also indexed positions $(N^A, \upsilon) \geq_{ind} f$ with ind being one of: (i) a step function, or (ii) a vector of states, or (iii) a rank in $[m]$. Usually we will omit the superscript A from N^A. The rules of the game are given in Figure 6. Eve chooses a transition from positions of the form $(a, \upsilon) \geq f$, and a step function g from positions of the form $(NK, \upsilon) \geq f$. Adam chooses a successor from positions of the form $(NK, \upsilon) \geq_g f \mapsto q$. The rank of a node $(N, \upsilon) \geq \vec{h} \mapsto q$ is the rank of q in $\mathcal{A}$. The rank of a node $(N, \upsilon) \geq_r \vec{h} \mapsto q$ is r, while the rank of a node labelled $(N, \upsilon) \geq_g \vec{h} \mapsto q$ is 0 when g is a step function. Eve wins an infinite play when the sequence of ranks on the play satisfies the parity condition. She also wins when she reaches a position of one of the two forms:

- $(a,\upsilon) \geq_{q_1,\ldots,q_k} h_1 \mapsto \ldots \mapsto h_k \mapsto q$ with $(q_1,\ldots,q_k) \in \delta(q,a)$, and $q_i \in (h_i\restriction_{rk(q)})^\partial$ for $i = 1,\ldots,k$.
- $(x,\upsilon) \geq \vec{h} \mapsto q$ iff $q \in (\upsilon(x))^\partial(\vec{h})$.

Otherwise the play is winning for Adam.

$$
\begin{aligned}
(a,\upsilon) \geq f &\longrightarrow (a,\upsilon) \geq_{q_1,\ldots,q_k} f \\
(Y\mathbf{x}.N,\upsilon) \geq f &\longrightarrow (N,\upsilon) \geq f \\
(\mathbf{x},\upsilon) \geq f &\longrightarrow (term(\mathbf{x}),\upsilon) \geq f \\
(\lambda x.M,\upsilon) \geq f_0 \mapsto f_1 \mapsto \ldots \mapsto f_k \mapsto q &\longrightarrow \\
(M,\upsilon[f_0/x]) \geq f_1 \mapsto \ldots \mapsto f_k \mapsto q & \\
(NK,\upsilon) \geq f &\longrightarrow (NK,\upsilon) \geq_g f \qquad \text{for every } g \in \mathcal{R}_B \\
& \qquad \text{with } B \text{ the type of } K \\
(NK,\upsilon) \geq_g \vec{f} \mapsto q &\longrightarrow (N,\upsilon) \geq g\restriction_q \mapsto \vec{f} \mapsto q \\
(NK,\upsilon) \geq_g \vec{f} \mapsto q &\longrightarrow (K,\upsilon) \geq_{r'} \vec{h}\restriction_{q'} \mapsto q' \\
& \qquad \text{for every } (q',r') \in g(\vec{h}\restriction_{q'}) \\
(K,\upsilon) \geq_r f &\longrightarrow (K,\upsilon\restriction_r) \geq f
\end{aligned}
$$

Figure 6 Rules of the game $\mathcal{G}(\mathcal{A})$.

Remark: The operations $g\restriction_q$ and $\vec{h}\restriction_{q'}$ in the definition of the game ensure that we use only step functions that are stratified. Observe that the clauses for winning positions in the game correspond directly to the clauses in the semantics.

Remark: While the game $\mathcal{G}(\mathcal{A})$ is infinite, for every position of the game there is a finite number of positions reachable from it. So for a fixed term the game is a finite parity game.

The next theorem says that our semantics characterizes the winning positions in this game. We need to do a small syntactic adjustment. Games use $Y\mathbf{x}.N$ notation while in the model construction we consider Y as a constant. In order to make things match let M^* be a term resulting from replacing all constants Y by a term $\lambda z.Y\mathbf{x}.z\mathbf{x}$.

▶ **Theorem 19.** *For every state $q \in Q$, term M, valuation υ, and a sequence of arguments $\vec{f}$ of appropriate types:*

$$q \in [\![M,\upsilon]\!]\vec{f} \text{ iff Eve wins from the position } (M^*,\upsilon) \geq \vec{f} \to q \text{ in } \mathcal{G}(\mathcal{A})$$

Proof. The proof is by induction on the size of M. The case of Y is treated separately in Proposition 20 below. Among the remaining cases only application is complicated. We present the left to right direction.

Take some $q \in [\![NK,\upsilon]\!]\vec{f}$. We need to show how Eve can win from $(NK,\upsilon) \geq \vec{f} \mapsto q$. A good strategy for Eve is to choose $g = \langle\!\langle K,\upsilon \rangle\!\rangle = (\bigvee_{r=0}^{m} [\![K,\upsilon\restriction_r]\!] \cdot r)$ as this will make the next (two) steps of the game $\mathcal{G}(\mathcal{A})$ give:

$$
\begin{array}{ccc}
 & (NK,\upsilon) \geq_g \vec{f} \to q & \\
\swarrow & & \searrow \text{ for all } \vec{h} \text{ and } (q',r') \in g(\vec{h}\restriction_{q'}) \\
(N,\theta) \geq g\restriction_q \to \vec{f} \to q & & (K,\upsilon\restriction_{r'}) \geq \vec{h}\restriction_{q'} \to q'
\end{array}
$$

The definition of $[\![NK, v]\!]$ tells us that $q \in [\![N, v]\!] g\vec{f}$. By the (strat) property from Lemma 9 we have $q \in [\![N, v]\!] g\restriction_q \vec{f}\restriction_q$. Then, since $g\restriction_q = (g\restriction_q)\restriction_q$ we can use once again Lemma 9 to obtain $q \in [\![N, v]\!] g\restriction_q \vec{f}$. So the position in the left branch is winning by induction hypothesis.

The statement $(q', r') \in g(\vec{h}\restriction_{q'})$ is by definition of g just $q' \in [\![K, v\restriction_{r'}]\!]\vec{h}\restriction_{q'}$. By induction hypothesis, this means that every position of the form in the right branch is winning. This concludes the case of application. ◀

▶ Proposition 20. For every $q \in Q$, type A, $f \in \mathcal{R}_{A\to A}$, and $\vec{h}$ of appropriate types:

$q \in [\![Y^A, \emptyset]\!] f\vec{h}$ iff Eve wins from the position $(\lambda z.Y\mathbf{x}.z\mathbf{x}, \emptyset) \geq f \mapsto \vec{h} \mapsto q$ in $\mathcal{G}(\mathcal{A})$.

The proof of this proposition makes the use of the fixpoint characterization of the set winning positions in a parity game [5, 26]. We extract the relevant part of $\mathcal{G}(\mathcal{A})$ and show some kind of equivalence between the fixpoint formula defining Y and the one characterizing the winning condition.

B.2 Winning in $\mathcal{G}(\mathcal{A})$ versus acceptance by $\mathcal{A}$

The missing piece is the following equivalence:

▶ Proposition 21. For every closed term M of type o, and every state q of the automaton $\mathcal{A}$:

$(M, \emptyset) \geq q$ is winning in $\mathcal{G}(\mathcal{A})$ iff $BT(M)$ is accepted by $\mathcal{A}$ from q.

This proposition is proved in two steps. First, we define a more liberal game $\mathcal{G}^{st}(\mathcal{A})$ and show the proposition for this more liberal game. Here we can reuse already known results. Next, we show the equivalence of $\mathcal{G}^{st}(\mathcal{A})$ and $\mathcal{G}(\mathcal{A})$.

The game $\mathcal{G}^{st}(\mathcal{A})$ is easier to win for Eve since she is allowed to use a weaker notion of residual: the requirement for monotonicity is dropped but the requirement to satisfy (strat) remains.

▶ **Definition 22.** The set $\mathcal{R}^{st}_A$ of *stratified residuals of type* A is defined by induction on A:

$\mathcal{R}^{st}_o = \mathcal{P}(\{(q, r) : rk(q) \leq r \leq m\})$
$\mathcal{R}^{st}_{A\to B} = \{R : \mathcal{R}^{st}_A \to \mathcal{R}^{st}_B : R \text{ satisfies (strat) condition}\}$

The game $\mathcal{G}^{st}(\mathcal{A})$ is as $\mathcal{G}(\mathcal{A})$ but where Eve can use step functions from $\{\mathcal{R}^{st}_A\}_{A\in types}$, and not just form $\{\mathcal{R}_A\}_{A\in types}$ as it is the case in $\mathcal{G}(\mathcal{A})$.

The following theorem linking the game $\mathcal{G}^{st}(\mathcal{A})$ and acceptance by $\mathcal{A}$ is the main reason for introducing $\mathcal{G}^{st}(\mathcal{A})$ as the intermediate step of the whole argument.

▶ **Theorem 23.** *For every closed term M of type o: the automaton $\mathcal{A}$ accepts $BT(M)$ from a state q iff the position $(M, \emptyset) \geq q$ is winning for Eve in $\mathcal{G}^{st}(\mathcal{A})$.*

The proof of this theorem is a refinement of the argument from [21] where the same characterization is proved for a variant of the game $\mathcal{G}^{st}(\mathcal{A})$; the main difference being the (strat) requirement. To take (strat) into account, it is enough to observe that the strategies constructed in the proof in op. cit. use residuals satisfying this property.

In the light of the above theorem, Proposition 21 is implied by the following proposition comparing games $\mathcal{G}(\mathcal{A})$ and $\mathcal{G}^{st}(\mathcal{A})$.

▶ Proposition 24. For every closed term M of type o, and every state q of $\mathcal{A}$: $(M, \emptyset) \geq q$ is winning in $\mathcal{G}(\mathcal{A})$ iff it is winning in $\mathcal{G}^{st}(\mathcal{A})$.

In the proof we essentially show how to find for every stratified residual an "equally good" monotone and stratified residual. While the construction is quite natural, the details are bit heavy. The main part is to show how to transfer a strategy in $\mathcal{G}^{st}(\mathcal{A})$ to a more constrained setting of $\mathcal{G}(\mathcal{A})$. For this we first associate to every stratified residual a monotone stratified residual. For $R_0 \in \mathcal{R}^{st}_o$, $R_1 \in \mathcal{R}^{st}_{A\to B}$ and $h \in \mathcal{R}^{st}_A$ we put:

$$\begin{aligned} mon(R_0) =& R_0 \\ mon(R_1)(h) =& \bigvee\{mon(R_1(S)) : mon(S) \leq h\} \end{aligned}$$

Recall that $\mathcal{R}_A$ is a complete lattice for every type A, so the join in the above definition exists. Actually it requires some calculation to show that $mon(R)$ is indeed in $\mathcal{R}$. This done, we can prove the announced transfer of strategies, and hence also Proposition 24.

▶ **Lemma 25.** *If $(M,\theta) \geq \vec{S} \mapsto q$ is a winning position for Eve in $\mathcal{G}^{st}(\mathcal{A})$ then for every $\upsilon \geq mon(\theta)$, and every $\vec{f} \geq mon(\vec{S})$, the position $(M,\upsilon) \geq \vec{f} \mapsto q$ is winning in $\mathcal{G}(\mathcal{A})$.*

C Omitted proofs from Section B

Theorem 19 For every state $q \in Q$, term M, valuation υ, and a sequence of arguments $\vec{f}$ of appropriate types:

$q \in [\![M,\upsilon]\!]\vec{f}$ iff Eve wins from the position $(M^*,\upsilon) \geq \vec{f} \to q$ in $\mathcal{G}(\mathcal{A})$

Proof. In the main text we have only given a part of this proof. Here we have added the missing argument.

The proof is by induction on the size of M. The case of Y is treated separately in Proposition 20 below. Among the other cases only the application is not direct. So we concentrate on that particular case. We first present left to right implication.

To examine the case where $M = NK$, take some $q \in [\![NK,\upsilon]\!]\vec{f}$. We need to show how Eve can win from $(NK,\upsilon) \geq \vec{f} \mapsto q$. A good strategy for Eve is to choose $g = \langle\!\langle K,\upsilon\rangle\!\rangle = (\bigvee_{r=0}^{m} [\![K,\upsilon\restriction_r]\!] \cdot r)$. Then in the following step the play in $\mathcal{G}(\mathcal{A})$ looks as follows:

$$\begin{array}{ccc} & (NK,\upsilon) \geq_g \vec{f} \to q & \\ \swarrow & & \searrow \text{ for all } \vec{h} \text{ and } (q',r') \in g(\vec{h}\restriction_{q'}) \\ (N,\theta) \geq g\restriction_q \to \vec{f} \to q & & (K,\upsilon\restriction_{r'}) \geq \vec{h}\restriction_{q'} \to q' \end{array}$$

The definition of $[\![NK,\upsilon]\!]$ tells us that $q \in [\![N,\upsilon]\!]g\vec{f}$. By the (`strat`) property from Lemma 9 we have $q \in [\![N,\upsilon]\!]g\restriction_q\vec{f}\restriction_q$. Then, since $g\restriction_q = (g\restriction_q)\restriction_q$ we can use once again Lemma 9 to obtain $q \in [\![N,\upsilon]\!]g\restriction_q\vec{f}$. So the position in the left branch is winning by induction hypothesis.

The statement $(q',r') \in g(\vec{h}\restriction_{q'})$ is by definition of g just $q' \in [\![K,\upsilon\restriction_{r'}]\!]\vec{h}\restriction_{q'}$. By induction hypothesis, this means that every position of the form in the right branch is winning. This concludes the case of application.

For the right to left implication suppose that $(NK,\upsilon) \geq \vec{f} \to q$ is winning. In this case Eve can choose some g and the play will develop as in the picture above.

Looking at the left branch, the induction hypothesis gives us $q \in [\![N,\upsilon]\!]g\restriction_q\vec{f}$. By the same reasoning as above we obtain $q \in [\![N,\upsilon]\!]g\vec{f}$ from Lemma 9. The right branch gives us $q' \in [\![K,\upsilon\restriction_{r'}]\!]\vec{h}\restriction_{q'}$, or writing it differently $(q',r') \in (\bigvee_{r=0}^{m} [\![K,\upsilon\restriction_r]\!] \cdot r)\vec{h}\restriction_{q'} = \langle\!\langle K,\upsilon\rangle\!\rangle\vec{h}\restriction_r$. As

$\langle\!\langle K, \upsilon \rangle\!\rangle$ satisfies (`strat`) we can conclude that $(q', r') \in \langle\!\langle K, \upsilon \rangle\!\rangle \vec{h}$. Since this holds for all $\vec{h}$ and all $(q', r') \in g(\vec{h})$ we obtain $g \leq \langle\!\langle K, \upsilon \rangle\!\rangle$. By monotonicity we get $q \in [\![N, \upsilon]\!] \langle\!\langle K, \upsilon \rangle\!\rangle \vec{f}$. So $q \in [\![NK, \upsilon]\!] \vec{f}$ as required. ◀

C.1 Correctness of the fixpoint

This rest of this subsection is devoted to the proof of the following proposition.

Proposition 20 For every $q \in Q$, type A, $f \in \mathcal{R}_{A \to A}$, and $\vec{h}$ of appropriate types:

$q \in [\![Y^A, \emptyset]\!] f \vec{h}$ iff Eve wins from the position $(\lambda z.Yx.zx, \emptyset) \geq f \mapsto \vec{h} \mapsto q$ in $\mathcal{G}(\mathcal{A})$.

We fix a type $A = \vec{B} \to o$, and $f \in \mathcal{R}_{A \to A}$. The first, rather cosmetic, step is to define a new game that is equivalent to $\mathcal{G}^{ms}(\mathcal{A})$ for the positions we are interested in.

▶ **Definition 26.** For a given A and f we define the game G_A^{Yf}. The positions for Eve in G_A^{Yf} are $(q, l, \vec{h})$ where $q \in Q$ is a state, $l \in [m]$ is a rank, and $\vec{h} \in \mathcal{R}_{\vec{B}}$ is a list of arguments of suitable types. The positions of Adam are (g, l) where $g \in \mathcal{R}_A$ and $l \in [m]$. The moves are as follows:

- from $(q, l, \vec{h})$ there is a move to every (g, l) such that $q \in (f\!\downarrow_l)^\partial(g, \vec{h})$. This transition has rank 0.
- from (g, l) there is a move to $(q', \max(l, r'), \vec{h'}\!\downarrow_{q'})$ for every $\vec{h'}$ and every $(q', r') \in g(\vec{h'}\!\downarrow_{q'})$. This transition has rank r'.

The winning condition in G_A^{Yf} is the parity condition given by the ranks on the transitions.

By direct examination of the definitions we obtain

▶ **Lemma 27.** *$(q, l, \vec{h})$ is winning for Eve in G_A^{Yf} iff $(\lambda z.Y(\lambda x.zx), \emptyset) \geq f\!\downarrow_l \to \vec{h} \to q$ is winning for Eve in $\mathcal{G}(\mathcal{A})$.*

In view of this lemma, in order to prove Proposition 20 we can work with the game G_A^{Yf}. The main advantage of this game is its simple structure. The winning positions in a parity game are characterized by a fixpoint formula, and the simpler the game the simpler the formula. We consider the game G_A^{Yf} as a transition system with positions as states and moves as transitions. Every transition has a label that is its rank. We will use a notation borrowed from modal logic. For a set of positions V of the game we write $\langle 0 \rangle V$ for the set of positions having a transition of rank 0 leading to V. Dually, $[i]V$ is the set of positions from which every transition of rank i leads to a state from V. With this view, the set of winning positions for Eve is the set of states satisfying the formula (we suppose that m is odd)

$$\theta_{win} = \mu X_m.\nu X_{m-1} \ldots \mu X_1.\nu X_0.\ \langle 0 \rangle \left(\bigwedge_{i=0}^{m} [i] X_i \right) \tag{1}$$

Intuitively the first $\langle 0 \rangle$ represents the choice of Eve, and then $[i]$ represent the choices of Adam, the rank of transition is reflected in the variable used.

Recall that the semantics of Y in the model is

$$[\![Y^A, \upsilon]\!] f = \mathrm{fix}(f, 0) \quad \text{where for } l = 0, \ldots, m \text{ we have}$$

$$\mathrm{fix}(f, l) = \sigma g_l. \ldots \mu g_1.\nu g_0.\ (f\!\downarrow_l)^\partial \big(\bigvee_{i=0}^{l} g_i \cdot i \vee \bigvee_{i=l+1}^{m} \mathrm{fix}(f, i) \cdot i \big)$$

With these preparations we have reduced Proposition 20 to the following lemma.

▶ **Lemma 28.** *For every l:* $\mathrm{fix}(f,l) \geq \vec{h} \mapsto \{q\}$ *iff* $(q,l,\vec{h})$ *is in* θ_{win}.

Indeed, as the semantics of Y is $\mathrm{fix}(f,0)$ we are done since by Lemma 27 winning from $(q,0,\vec{h})$ in G_A^{Yf} is the same as winning from $(\lambda z.Yx.zx,\emptyset) \geq f \mapsto \vec{h} \mapsto q$ in in $\mathcal{G}(\mathcal{A})$.

Given a set V of positions that belong to Eve in G_A^{Yf}, we define $[V,k]$ to be the function: $\bigvee\{\vec{h} \mapsto q : (q,k,\vec{h}) \in V\}$. With this notation, the equivalence characterizing $\mathrm{fix}(f,l)$ can be rephrased as $\mathrm{fix}(f,l) = [\theta_{win},l]$. We will prove this statement below (Corollary 34).

We say that V is *saturated*, when if $(q,k,\vec{h})$ is in V, then for every $\vec{h}' \geq \vec{h}$, $(q,k,\vec{h}')$ is in V. We show that the property of being saturated is preserved by all the operations we are interested in.

▶ **Lemma 29.** *Given a family of $(V_i)_{i\in I}$ of saturated sets of positions, $\bigcap_{i\in I} V_i$ and $\bigcup_{i\in I} V_i$ are also saturated.*

▶ **Lemma 30.** *Given arbitrary sets of positions $V_1, \ldots, V_m$ in G_A^{Yf}, $\langle 0\rangle(\bigwedge_{i=0}^m [i]V_i)$ is a saturated set.*

Proof. If $(q,k,\vec{h})$ is in $\langle 0\rangle(\bigwedge_{i=0}^m [i]V_i)$, this means that there is g so that $q \in (f\lfloor_k)^\partial(g,\vec{h})$ and for every $\vec{h}'$, (q,i) in $g(\vec{h}')$ implies that $(q',k,\vec{h}')$ is in V_i. Take $\vec{h}' \geq \vec{h}$, then by monotonicity of $(f\lfloor_k)^\partial$, $q \in (f\lfloor_k)^\partial(g,\vec{h}')$ which implies that $(q,k,\vec{h}')$ is in $\langle 0\rangle(\bigwedge_{i=0}^m [i]V_i)$. ◀

▶ **Lemma 31.** *Fix $0 \leq k \leq m$, for all sets $V_k, \ldots, V_m$, the set*

$$\sigma X_{k-1} \ldots \mu X_1.\nu X_0.\langle 0\rangle(\bigwedge_{i=0}^{k-1} [i]X_i \wedge \bigwedge_{i=k}^{m} [i]V_i)$$

is saturated.

Proof. We proceed by induction on k. The case where $k = 0$ is treated in the previous lemma. The induction case is a simple consequence of Lemma 29 and of the definition of least and greatest fixpoints as unions and intersections, respectively. ◀

The next lemma is the core of the proof of Proposition 20. It gives a correspondence between the inner operation from the definition of θ_{win} and the ones from the definition of $\mathrm{fix}(f,l)$.

▶ **Lemma 32.** *For all $V_0,\ldots,V_m$ saturated sets of positions in the game G_A^{Yf}, for every state q and vector of elements $\vec{h}$ of appropriate types:*

$$(f\lfloor_l)^\partial\left(\bigvee_{i=0}^{l}[V_i,l]\cdot i \vee \bigvee_{i=l+1}^{m}[V_i,i]\cdot i\right) \geq \vec{h} \mapsto \{q\}$$

iff

$$(q,l,\vec{h}) \in \langle 0\rangle\left(\bigwedge_{i=0}^{m}[i]V_i\right) .$$

Proof. Let $g = \bigvee_{i=0}^{l}[V_i,l]\cdot i \vee \bigvee_{i=l+1}^{m}[V_i,i]\cdot i$ in the course of this proof.

If we suppose that $(f\lfloor_l)^\partial(g) \geq \vec{h} \mapsto \{q\}$, then by definition, there is a transition from $(q,l,\vec{h})$ to (g,l) in G_A^{Yf}. We need to show that $(g,l) \in (\bigwedge_{i=0}^m [i]V_i)$. Given $\vec{h}'$ let (q',i) be in $g(\vec{h}'\lfloor_{q'})$. There are two cases. The first is when $i \leq l$. We need to prove that $(q',\max(l,i),\vec{h}'\lfloor_{q'}) = (q',l,\vec{h}'\lfloor_{q'})$ is in V_i. As, (q',i) is in $g(\vec{h}'\lfloor_{q'})$, it must be the case that q' is in $[V_i,l](\vec{h}'\lfloor_{q'})$. From the definition of $[V_i,l]$, and the fact that V_i is saturated, we obtain

that $(q', l, \vec{h}'\lfloor_{q'})$ is in V_i. The second case is when $i > l$. Since (q', i) belongs to $g(\vec{h}'\lfloor_{q'})$, we must have q' in $[V_i, i](\vec{h}'\lfloor_{q'})$. By definition this means that $(q', i, \vec{h}'\lfloor_{q'}) = (q', \max(l, i), \vec{h}'\lfloor_{q'})$ is in V_i as expected.

For the converse direction, suppose $(q, l, \vec{h})$ is in $\langle 0\rangle\,(\bigwedge_{i=0}^{m}[i]V_i)$, there must be (g', l) so that:

1. q is in $f\lfloor_l^{\partial}(g', \vec{h})$, and
2. for every $\vec{h}'$, and every (q', i) in $g'(\vec{h}'\lfloor_{q'})$, $(q', \max(l, i), \vec{h}'\lfloor_{q'})$ is in V_i.

We need to prove $g' \leq g$. For this it is sufficient to show that for every $\vec{h}'$, $(q', i) \in g'(\vec{h}')$ implies $(q', i) \in g(\vec{h}')$. Since both g' and g satisfy (`strat`) it is sufficient to show this for $\vec{h}'$ such that $\vec{h}'\lfloor_q = \vec{h}'$. So we fix such $\vec{h}'$ and take (q', i) in $g'(\vec{h}')$. In the case where $i \leq l$, we can remark that if $(q', \max(l, i), \vec{h}') = (q', l, \vec{h}')$ is in V_i then $q' \in [V_i, l](\vec{h}')$ and therefore $(q', i) \in g(\vec{h}')$. In the case where $i > l$, if $(q', \max(l, i), \vec{h}') = (q', i, \vec{h}')$ is in V_i, then $q' \in [V_i, i](\vec{h}')$ and thus $(q', i) \in g(\vec{h}')$. So in every case we have seen $(q', i) \in g'(\vec{h}')$ implies $(q', i) \in g(\vec{h}')$ and thus $g' \leq g$. As $(f\lfloor_l)^{\partial}$ is monotone, we obtain $q' \in (f\lfloor_l)^{\partial}(g, \vec{h})$. ◀

We then extend the previous lemma to formulas with fixpoints.

▶ **Lemma 33.** *For every k and l verifying $0 \leq k \leq l \leq m$, and for all saturated sets V_k, …, V_m, we have:*

$$\sigma g_{k-1}. \ldots \mu g_1 \nu g_0.$$
$$(f\lfloor_l)^{\partial}\Big(\bigvee_{i=0}^{k-1} g_i \cdot i \vee \bigvee_{i=k}^{l} [V_i, l] \cdot i \vee \bigvee_{i=l+1}^{m} [V_i, i] \cdot i\Big) \geq \vec{h} \mapsto \{q\}$$

iff

$$(q, l, \vec{h}) \in \sigma X_{k-1}. \ldots \mu X_1 \nu X_0. \langle 0\rangle\Big(\bigwedge_{i=0}^{k-1}[i]X_i \wedge \bigwedge_{i=k}^{m}[i]V_i\Big)\ .$$

Proof. We fix $l \in [m]$ and we prove the lemma by induction on k.

The proof for $k = 0$ starts from the statement of the previous lemma. The induction step uses the unrolling of the greatest/least fixpoint definition. This case also relies on Lemma 31 so as to ensure that at each iteration of a fixpoint, the hypothesis that the new set V_k is saturated is verified. ◀

▶ **Corollary 34.** *For every $l \in [m]$, $\mathrm{fix}(f, l) = [\theta_{win}, l]$.*

Proof. We proceed by induction on $m - l$. The case where $l = m$, is an immediate consequence of the previous lemma when taking $k = m$.

Now, for the inductive case, we assume that for every $i > l$, $\mathrm{fix}(f, i) = [\theta_{win}, i]$. As $\mathrm{fix}(f, i)$ is a monotone function, this implies that $\theta^i_{win} = \{(q, i, \vec{h}) \in \theta_{win} : \text{for some } q \text{ and } \vec{h}\}$ is a saturated set. Then, the previous lemma allows us to conclude that $q \in \mathrm{fix}(f, l)(\vec{h})$ iff $(q, l, \vec{h})$ is in $\sigma X_l. \ldots \mu X_1. \nu X_0. \langle 0\rangle(\bigwedge_{i=0}^{l}[i]X_i \wedge \bigwedge_{i=l+1}^{m}[i]\theta^i_{win})$. As moreover, the latter set is just θ_{win} we obtain $\mathrm{fix}(f, l) = [\theta_{win}, l]$. ◀

As noted above this Corollary is a reformulation of Lemma 28. We have already shown that the lemma implies in turn Proposition 20.

C.2 Winning in $\mathcal{G}^{st}(\mathcal{A})$ versus acceptance

In this subsection we present the proof the following theorem

Theorem 23 For every closed term M of type o: the automaton $\mathcal{A}$ accepts $BT(M)$ from state q iff the position $(M,\emptyset) \geq q$ is winning for Eve in $\mathcal{G}^{st}(\mathcal{A})$.

The variant of this theorem is proved in already in [21]. We will not recall all the material from that paper. In particular we will omit the definition of the Krivine machine.

C.2.1 Game $\mathcal{K}(\mathcal{A}, M)$

We now give the definition of $RT(\mathcal{A}, M)$, the runs of the automaton $\mathcal{A}$ on the graph of configurations of the Krivine Machine computing $BT(M)$. The actual runs of $\mathcal{A}$ on $BT(M)$ can easily be read off $RT(\mathcal{A}, M)$.

The labels of the tree $RT(\mathcal{A}, M)$ will be of the form $(N,\rho) \geq C$ where N is a term, ρ is an environment, and C is a *closure expression*. The latter is either just a state q or has the form $(u, K, \rho') \mapsto C$ where u is a node of $RT(\mathcal{A}, M)$, and (K, ρ') is a closure. We will also have labels with indices $(N,\rho) \geq_{ind} C$, where ind is a pair of states or a node of $RT(\mathcal{A}, M)$.

▶ **Definition 35.** For a given closed λY-term M of type o , and a parity automaton $\mathcal{A}$ we define the tree of runs $RT(\mathcal{A}, M)$ of $\mathcal{A}$ on the execution tree of the Krivine Machine on M:

1. The root of the tree is labelled with $(M,\emptyset) \geq q^0$.
2. A node labelled $(a,\rho) \geq C$ has a successor $(a,\rho) \geq_{q_1,\dots,q_k} C$ for every $(q_1,\dots,q_k) \in \delta(q,a)$.
3. A node labelled $(a,\rho) \geq_{q_1,\dots,q_k} C_1 \mapsto \cdots \mapsto C_k \mapsto q$, where $C_i = (u_i, N_i, \rho_i)$, has successors $(N_i,\rho_i) \geq_{u_i} q_i$ for $i = 1,\dots,k$.
4. A node labelled $(\lambda x.N,\rho) \geq C \mapsto D$ has a unique successor labelled $(N, \rho[C/x]) \geq D$.
5. A node $(Y\mathbf{x}.N,\rho) \geq C$ has a unique successor $(N,\rho) \geq C$.
6. A node labelled $(\mathbf{x},\rho) \geq C$, for $\mathbf{x}$ a recursive variable, has a unique successor $(term(\mathbf{x}),\emptyset) \geq C$.
7. A node u labelled $(NK,\rho) \geq C$ has a unique successor labelled $(N,\rho) \geq (u, K, \rho) \mapsto C$. We say that here a *u-closure is created*.
8. A node labelled $(x,\rho) \geq C$, for x a λ-variable and $\rho(x) = (u', N, \rho')$, has a unique successor labelled $(N,\rho') \geq_{u'} C$.
9. A node labelled $(N,\rho) \geq_u C$ has a unique successor labelled $(N,\rho) \geq C$.

We will say that in the nodes of the form $(N,\rho) \geq_u C$ the closure (u, N, ρ) is *used*.

The definition is as expected but for the fact that in the rule for application we store the current node in the closure. When we use the closure in the variable rule or constant rule (rules 8 and 3), the stored node does not influence the result. The stored node is just used to detect what is exactly the closure that we are using. This will be important in the proof.

Notice also that the rules 2,3,4 rely on the typing properties of the Krivine machine ensured by the definition of its configurations (cf. [21]). Indeed, when the machine reaches a configuration of the form $(a,\rho) \geq C$ then, since we are working with a tree signature, a is of type $o^k \to o$ for some k. In consequence, C is of the form $D_1 \mapsto \cdots \mapsto D_k \mapsto q$ with $D_1,\dots,D_k$ are k closures of type o. The environment ρ plays no role in such a configuration as a is a constant. Also from the typing invariant we get that, when the machine is in a configuration like $(\lambda x.N,\rho) \geq C$ then C is of the form $C' \mapsto D$.

▶ **Definition 36.** We use the tree $RT(\mathcal{A}, M)$ to define a game between two players: Eve chooses a successor in nodes of the form $(a,\rho) \geq C$, and Adam in nodes $(a,\rho) \geq_{q_1,\dots,q_k} C$.

The parity rank of a node $(N,\rho) \geq D \to q$ is $rk(q)$. We can use max parity condition to decide who wins an infinite play. Let us call the resulting game $\mathcal{K}(\mathcal{A}, M)$.

The following has been proved in [21].

▶ Proposition 37. For every parity automaton $\mathcal{A}$ and concrete canonical term M. Eve has a strategy from the root position in $\mathcal{K}(\mathcal{A}, M)$ iff $\mathcal{A}$ accepts $BT(M)$.

The only interesting point to observe is that the above definition is consistent with our assumption that the Böhm tree consisting of the root labelled by Ω is accepted from states of even rank and rejected from states of odd rank.

The above proposition reduced deciding whether $BT(M)$ is accepted by $\mathcal{A}$ is reduced to deciding who has a winning strategy from the root of $\mathcal{K}(\mathcal{A}, M)$. We will introduce a "smaller" game $G(\mathcal{A}, M)$, and show that the winner in the two games is the same. The game $\mathcal{G}^{st}(\mathcal{A})$ is the union of the finite games $G(\mathcal{A}, M)$.

C.2.2 Game $\mathcal{G}(\mathcal{A}, M)$

The positions of the game are of the form $(N,\theta) \geq S$ where N is a subterm of M, θ is a function assigning residuals to variables of M, and S is a *residual expression*. The latter is either a state of $\mathcal{A}$, or $R \mapsto S$ where R is a residual and S a residual expression. We will also have positions with indices $(N,\theta) \geq_{ind} S$, where ind is a pair of states, a rank, or a residual.

▶ **Definition 38.** The game $G(\mathcal{A}, M)$ is as follows:

1. The initial position is $(M, \emptyset) \geq q^0$
2. A node $(a,\theta) \geq S$ has a successor $(a,\theta) \geq_{q_1,\ldots,q_k} S$ for every $(q_1,\ldots,q_k) \in \delta(q,a)$.
3. A node $(Y\mathbf{x}.N,\theta) \geq S$ has a successor $(N,\theta) \geq S$.
4. A node $(\mathbf{x},\theta) \geq S$, for $\mathbf{x}$ a recursive variable, has a successor $(term(\mathbf{x}),\theta) \geq S$.
5. A node $(NK,\theta) \geq S$ has a successor $(NK,\theta) \geq_R S$ for every R residual of the type of K.
6. A node $(NK,\theta) \geq_R \vec{S} \mapsto q$ has two types of successors
   - one successor $(N,\theta) \geq R|_q \mapsto \vec{S} \mapsto q$, and
   - for every $\vec{P}$ and every $(q',r') \in R(\vec{P})$ with $r' \geq \max(rk(q'), rk(q))$ a successor $(K,\theta) \geq_{r'} \vec{P} \mapsto q'$.
7. A node $(K,\theta) \geq_{r'} \vec{P} \mapsto q'$ has a unique successor $(K, \theta|_{r'}) \geq \vec{P} \mapsto q'$.

The rank of a node labelled $(N,\theta) \geq \vec{S} \mapsto q$ is the rank of q. The rank of a node labelled $(N,\theta) \geq_r \vec{S} \mapsto q$ is r, while the rank of a node labelled $(N,\theta) \geq_R \vec{S} \mapsto q$ is 0 when R is a residual.

A position $(a,\theta) \geq_{q_1,\ldots,q_k} R_1 \mapsto \cdots \mapsto R_k \mapsto q$ is winning for Eve iff $(q_i, rk(q_i)) \in R_i|_{\max(rk(q),rk(q_i))}$ for $i = 1,\ldots,k$.

A position $(x,\theta) \geq \vec{S} \mapsto q$ is winning for Eve iff $(q, rk(q)) \in \theta(x)(\vec{S})$.

C.2.3 Residuals in $\mathcal{K}(\mathcal{A}, M)$

We here introduce the key notion of the proof, the notion of *residuals of nodes*. Given a subtree $\mathcal{T}$ of $\mathcal{K}(\mathcal{A}, M)$, i.e. a tree obtained from $\mathcal{K}(\mathcal{A}, M)$ by pruning some of its subtrees, we calculate the residuals $R_{\mathcal{T}}(u)$ and $res_{\mathcal{T}}(u,u')$ for some nodes and pair of nodes of $\mathcal{T}$. In particular, $\mathcal{T}$ may be taken as being a strategy of Eve or a strategy of Adam. When $\mathcal{T}$ is clear from the context we will simply write $R(u)$ and $res(u,u')$.

Recall that a node v in $\mathcal{K}(\mathcal{A}, M)$ is an application node when its label is of the form $(NK,\rho) \geq A$. In such node a closure (u, K, ρ) is created. We will define a residual $R(u)$ for

such a closure. Thanks to typing, this can be done by induction on the order of type. We also define a variation of this notion: a residual $R(u)$ seen from a node u', denoted $res(u,u')$. The two notions are the main technical tools used in the proof of the theorem.

Before giving a formal definition we will describe the assignment of residuals to nodes in concrete terms. We will need one simple abbreviation. If u is an ancestor of u' in $\mathcal{T}$ then we write $\max(u,u')$ for the maximal rank appearing on the path between u and u', including both ends.

Consider an application node u in $\mathcal{T}$. It means that u has a label of the form $(NK,\rho)\geq C$, and its unique successor has the label $(N,\rho)\geq(u,K,\rho)\mapsto C$. That is the closure (u,K,ρ) is created in u. We will look at all the places where this closure is used and summarize the information about them in $R(u)$. We will do this by induction on the type of K.

First, suppose that the closure, or equivalently the term K, is of type o. The residual $R(u)$ is a subset of $Q\times[d]$. It contains all pairs $(q',r')\in R(u)$ such that there is u' in $\mathcal{T}$ labelled $(K,\rho)\geq_u q'$ and $r'=\max(u,u')$. Observe that u' is necessarily a descendant of u.

For the induction step, suppose that K is of type $B_1\to\cdots\to B_k\to o$ and that we have already calculated residuals for all closures of types $B_1,\dots,B_k$. Consider a closure (u,K,ρ). This time $R(u)\in\mathcal{R}^{st}_{B_1\to\cdots\to B_k\to o}$. A node u' using the closure has the label of the form $(K,\rho)\geq_u C_1\mapsto\cdots\mapsto C_k\mapsto q'$ for some q' and $C_i=(u_i,N_i,\rho_i)$, for $i=1,\dots k$. We put

$$(q',\max(u,u'))\in R(u)(R(u_1)\downarrow_{\max(u_1,u')},\dots,R(u_k)\downarrow_{\max(u_k,u')})\ .$$

We now give a formal definition of $R(u)$. By structural induction on types it is easy to see that such an assignment of residuals exists and is unique for $\mathcal{T}$.

▶ **Definition 39** ($R(u)$ and $res(u,u_1)$)**.** Given $\mathcal{T}$ a subtree of $\mathcal{K}(\mathcal{A},M)$, we define a residual $R(u)$ for every application node u of $\mathcal{T}$.

For more clarity we will write $res(u,u_1)$ for $R(u)\downarrow_{\max(u,u_1)}$. For a closure (u,K,ρ) we define $res((u,K,\rho),u')=res(u,u')$. We then extend this operation to sequences of closures: $res(\vec{C},u')$ is a sequence of residuals obtained by applying $res(\cdot,u')$ to every element of $\vec{C}$.

Consider a closure (u,K,ρ) with K of type $A_1\to\cdots\to A_k\to o$. The residual $R(u)$ is in $\mathcal{R}^{st}_{A_1\to\cdots\to A_k\to o}$ and for every sequence of residuals $\vec{S}$ of appropriate types the set $R(u)(\vec{S})$ contains pairs $(q',\max(u,u'))$ for every node u' labelled by $(K,\rho)\geq_u\vec{C}\mapsto q'$ with $res(\vec{C},u')=\vec{S}$.

C.2.4 Transferring Eve's strategy from $\mathcal{K}(\mathcal{A},M)$ to $G(\mathcal{A},M)$

The invariant Will use positions in the game $\mathcal{K}(\mathcal{A},M)$ and the strategy σ as hints. The strategy in $G(\mathcal{A},M)$ will take a pair of positions (u,v) with u in $\mathcal{K}(\mathcal{A},M)$ and a v in $G(\mathcal{A},M)$. It will then give a new pair of positions (u',v') such that v' is a successor v, and u' is reachable from u using the strategy σ. Moreover, all visited pairs (u,v) will satisfy the following invariant:

> u is labelled by $(N,\rho)\geq\vec{C}\mapsto q$ and v is $(N,\theta)\geq\vec{R}\mapsto q$ with $\theta=res(\rho,u)$ and $\vec{R}=res(\vec{C},u)$.

The strategy. In $G(\mathcal{A},M)$ there are two kinds of nodes where Eve needs to decide which successor to choose: nodes with a constant, and nodes with an application. We show how Eve should play in order to preserve the invariant.

If the play reaches a node v with a constant in the label then the invariant tells us that we have an associated node u with the following situation

$$u : (a, \rho) \geq C_1 \mapsto \cdots \mapsto C_k \mapsto q \qquad \text{and} \qquad v : (a, \theta) \geq R_1 \mapsto \cdots \mapsto R_k \mapsto q,$$

Since u is a part of the winning strategy for Eve in $\mathcal{K}(\mathcal{A}, M)$, this node has a successor labelled $(a, \rho) \geq_{q_1,\ldots,q_k} C_1 \mapsto \cdots \mapsto C_k \to q$. It is clear that from v Eve can choose $(a, \theta) \geq_{q_1,\ldots,q_k} R_1 \mapsto \cdots \mapsto R_k \mapsto q$. We show that this position is winning for Eve.

By the invariant, we know that $R_i = res(C_i, u)$. Suppose $C_i = (u_i, N_i, \rho_i)$. Since u is a part of the winning strategy for Eve in $\mathcal{K}(\mathcal{A}, M)$, the successor of u has itself two successors labelled $(N_i, \rho_i) \geq_{u_i} q_i$ for $i = 1, \ldots, k$.

Looking at the definition of winning positions in $G(\mathcal{A}, M)$, we need to show $(q_i, rk(q_i)) \in R_i\!\downarrow_{\max(rk(q),rk(q_i))}$, for $i = 1, \ldots, k$. Definition of $R(u_i)$ gives $(q_i, \max(\max(u_i, u), rk(q_i))) \in R(u_i)$. Recall that $R_i = res(C_i, u) = R(u_i)\!\downarrow_{\max(u_i,u)}$. Since $rk(q) \leq \max(u_i, u)$, we obtain

$$(q_i, rk(q_i)) \in (R(u_i)\!\downarrow_{\max(u_i,u)})\!\downarrow_{\max(rk(q),rk(q_i))} = R_i\!\downarrow_{\max(rk(q),rk(q_i))}.$$

The second case is that of the application. In this case, the invariant tells us that we have

$$u : (NK, \rho) \geq \vec{C} \mapsto q \qquad \text{and} \qquad v : (NK, \theta) \geq \vec{S} \mapsto q$$

with $\theta = res(\rho, u)$ and $\vec{S} = res(\vec{C}, u)$.

In v Eve needs to choose a residual R and move to a node $(NK, \theta) \geq_R \vec{S} \mapsto q$. The strategy for Eve is to choose $R = R(u)$, the residual of the closure created at u in $\mathcal{K}(\mathcal{A}, M)$.

From the position $(NK, \theta) \geq_{R(u)} \vec{S} \mapsto q$ in $G(\mathcal{A}, M)$ Adam can choose:

1. either $(N, \theta) \geq R(u)\!\downarrow_q \mapsto \vec{S} \mapsto q$, or
2. $(K, \theta) \geq_{r'} \vec{P} \mapsto q'$ for some $\vec{P}$ and some $(q', r') \in R(u)(\vec{P})$ with $r' \geq \max(rk(q'), rk(q))$.

For each of these choices we need to find appropriate nodes in $\mathcal{K}(\mathcal{A}, M)$. The node u has a unique successor u' labelled by $(N, \rho) \geq (u, K, \rho) \mapsto \vec{C} \mapsto q$. So u' looks like a good choice for $(N, \theta) \geq R\!\downarrow_q \to \vec{S} \to q$. Indeed, $\theta = res(\rho, u') = res(\rho, u)$ is guaranteed by the invariant in u. From the invariant we also have $\vec{S} = res(\vec{C}, u') = res(\vec{C}, u)$. Finally $R(u)\!\downarrow_q = res(R(u), u')$, so the invariant is holds for u'.

We make here a small observation concerning ranks that will be importnat later. Observe that the rank of u and u' is just $rk(q)$. The rank of all positions met in the corresponding part of the play from v is $rk(q)$ too.

Let us consider now the second case, that is when Adam chooses $(K, \theta) \geq_{r'} \vec{P} \mapsto q'$ for some $\vec{P}$, q' and r'. This node has a unique successor $(K, \theta\!\downarrow_{r'}) \geq \vec{P} \mapsto q'$ and we want to find a node of $\mathcal{K}(\mathcal{A}, M)$ corresponding to it.

From $(q', r') \in R(u)(\vec{P})$ the definition of $R(u)$ tells us that there is a descendant $u^{\vec{P}}_{q',r'}$ of u labelled by $(K, \rho) \geq_u \vec{C} \mapsto q'$ with $res(\vec{C}, u^{\vec{P}}_{q',r'}) = \vec{P}$, and $r' = \max(u, u^{\vec{P}}_{q',r'})$. Node $u^{\vec{P}}_{q',r'}$ has a unique successor u' labelled $(K, \rho) \geq \vec{C} \mapsto q'$. We show that the invariant is satisfied for u' and $(K, \theta\!\downarrow_{r'}) \geq \vec{P} \mapsto q'$. We have $\vec{P} = res(\vec{C}, u^{\vec{P}}_{q',r'}) = res(\vec{C}, u')$. Concerning the environments, we know $\theta = res(\rho, u)$, since the invariant holds for (u, v). But then $res(\rho, u') = res(\rho, u^{\vec{P}}_{q',r'}) = res(\rho, u)\!\downarrow_{\max(u,u^{\vec{P}}_{q',r'})} = res(\rho, u)\!\downarrow_{r'} = \theta\!\downarrow_{r'}$, and we are done.

Similary, as above let us look at the rank of positions seen from u and v. The maximal rank seen from u to u' is r'. From v to $(K, \theta\!\downarrow_{r'}) \geq \vec{P} \mapsto q'$ we the positions of ranks $rk(q)$, r' and $rk(q')$. So on this side too the maximal rank seen is r'.

C.2.5 The strategy is winning

The initial position in $G(\mathcal{A}, M)$ is $(M, \emptyset) \geq q$. The root of $\mathcal{K}(\mathcal{A}, M)$ is labelled $(M, \emptyset) \geq q$. So clearly the invariant is satisfied for the pair of initial positions.

A direct examination of the rules shows that if (u, v) satisfy the invariant and node v has a unique successor then u has also the unique successor and this successor makes the invariant satisfied. In the definition of the strategy for Eve above we have shown how she can play in order to preserve the invariant in nodes with more than one successor.

In order to verify that the strategy is winning consider a play $v_0, v_1, \dots$ in $G(\mathcal{A}, M)$ with respect to this strategy. The strategy gives us the sequence of nodes in $\mathcal{K}(\mathcal{A}, M)$: $u_0, u_1, \dots$ such that the invariant holds for every pair (u_i, v_i). From the way we have constructed the strategy we get that u_{i+1} is a successor of u_i in all but one case appearing in the application rule. In that unique case u_{i+1} is a descendant of u_i, and the rank of v_{i+1} is the maximal rank of a state on the path from u_i to u_{i+1}. This means that the maximal rank appearing infinitely often on $v_0, v_1, \dots$ is the same as on $u_0, u_1, \dots$. So every infinite play respecting our strategy is winning, since $u_0, u_1, \dots$ is winning.

It remains to check what happens when a maximal play is finite. This happens when the last position v on the play does not have successors. We have two cases: either the term in v is a variable or a constant. The constant case has been dealt with when we have defined the strategy for Eve.

In the case where the term is a variable we are in the situation

$$u : (x, \rho) \geq \vec{C} \to q \qquad \text{iff} \qquad v : (x, \theta) \geq \vec{S} \to q$$

with $\vec{S} = \mathit{res}(\vec{C}, u)$ and $\theta(x) = \mathit{res}(\rho(x), u)$. We need to show that $(q, rk(q)) \in \theta(x)(\vec{S})$. Suppose $\rho(x) = (u', K, \rho')$, so $\theta(x) = R(u')\lfloor_{\max(u',u)}$.

In $\mathcal{K}(\mathcal{A}, M)$, the node u has a successor labelled by $(K, \rho') \geq_{u'} \vec{C} \to q$. By definition of $R(u')$ we have $(q, r') \in R(u')(\vec{S})$ where $r' = \max(u', u)$. So $(q, r') \in R(u')\lfloor_{r'}(\vec{S})$ which by definition of $\lfloor_r$ gives $(q, rk(q)) \in R(u')\lfloor_{r'}(\vec{S})$.

C.2.6 Transferring Adam's strategy from $\mathcal{K}(\mathcal{A}, M)$ to $G(\mathcal{A}, M)$

In order to formulate the invariant for the strategy we introduce complementarity predicate $\mathit{Comp}(R_1, R_2)$ between a pair of residuals:

- For $R_1, R_2 \in D_0$ we put $\mathit{Comp}(R_1, R_2)$ if $R_1 \cap R_2 = \emptyset$.
- For $R_1, R_2 \in D_A$ where $A = A_1 \to \dots \to A_k \to 0$ we put $\mathit{Comp}(R_1, R_2)$ if for all sequences $(R_{1,1}, \dots, R_{1,k}), (R_{2,1}, \dots, R_{2,k}) \in D_{A_1} \times \dots \times D_{A_k}$ such that $\mathit{Comp}(R_{1,i}, R_{2,i})$ for all $i = 1, \dots, k$ we get $R_1(R_{1,1}, \dots, R_{1,k}) \cap R_2(R_{2,1}, \dots, R_{2,k}) = \emptyset$.

As can be expected, for two closures (v, N, ρ) and (v', N, ρ') we say that the predicate $\mathit{Comp}((v, N, \rho), (v', N, \rho'))$ holds if $\mathit{Comp}(R(v), R(v'))$ is true. For two sequences of closures $\vec{S}_1, \vec{S}_2$ of the same length, $\mathit{Comp}(S, S')$ holds if the predicate holds for every coordinate. Finally, for two environments ρ, ρ' we write $\mathit{Comp}(\rho, \rho')$ if the two environments have the same domain and for every x, the predicate $\mathit{Comp}(\rho(x), \rho'(x))$ holds.

It is important to observe that Comp behaves well with respect to $\lfloor_r$ operation

▶ **Lemma 40.** *If $\mathit{Comp}(R_1, R_2)$ then also $\mathit{Comp}(R_1\lfloor_r, R_2\lfloor_r)$ for every rank r.*

The invariant The pairs (u, v) will satisfy

u is labelled by $(N, \rho) \geq \vec{C} \to q$, and v is $(N, \theta) \geq \vec{R} \to q$ where $\mathit{Comp}(\theta, \mathit{res}(\rho, u))$ and $\mathit{Comp}(\vec{R}, \mathit{res}(\vec{C}, u))$.

The strategy Adam makes a choice only in the application rule. In this case the invariant tells us that we have

$$u : (NK, \rho) \geq \vec{C} \mapsto q \qquad \text{and} \qquad v : (NK, \theta) \geq \vec{S} \mapsto q$$

with $\mathit{Comp}(\theta, \mathit{res}(\rho, u))$ and $\mathit{Comp}(\vec{S}, \mathit{res}(\vec{C}, u))$.

From node v Eve chooses some successor $(NK, \theta) \geq_R \vec{S} \mapsto q$, and then Adam can choose:

1. either $(N, \theta) \geq R\restriction_q \mapsto \vec{S} \mapsto q$, or
2. $(K, \theta) \geq_{r'} \vec{P} \mapsto q'$ for some $\vec{P}$ and some $(q', r') \in R(\vec{P})$ with $r' \geq \max(rk(q'), rk(q))$.

Adam should choose the node of the first type if $\mathit{Comp}(R, R(u))$ holds. Indeed the unique successor u' of u is labelled by $(N, \rho) \geq (u, K, \rho) \mapsto \vec{C} \mapsto q$. We have $\mathit{Comp}(\theta, \mathit{res}(\rho, u'))$ since $\mathit{res}(\rho, u') = \mathit{res}(\rho, u)$ and $\mathit{Comp}(\theta, \mathit{res}(\rho, u))$ holds by assumption. Similarly we have $\mathit{Comp}(\vec{S}, \mathit{res}(\vec{C}, u'))$. Finally, $\mathit{Comp}(R\restriction_q, \mathit{res}(u, u))$ holds since from $\mathit{Comp}(R, R(u))$ we can deduce $\mathit{Comp}(R\restriction_q, R(u)\restriction_q)$ by Lemma 40, and $R(u)\restriction_q = \mathit{res}(u, u)$ since q is the state in u.

If $\mathit{Comp}(R, R(u))$ does not hold then by the definition of this predicate there are $\vec{P}$ and $\vec{T}$ such that $\mathit{Comp}(\vec{P}, \vec{T})$ but $R(\vec{P}) \cap R(u)(\vec{T})$ is not empty. Let (q', r') be an element from this intersection. Adam should choose $(K, \theta) \geq_{r'} \vec{P} \mapsto q'$. Then the play will move to $(K, \theta\restriction_{r'}) \geq \vec{P} \mapsto q'$, and we need to find a corresponding node in $\mathcal{K}(\mathcal{A}, M)$ to restore the invariant.

Looking what $(q', r') \in R(u)(\vec{T})$ means we get that there is a descendant u' of u labelled by $(K, \rho) \geq_u \vec{D} \mapsto q'$ with $\vec{T} = \mathit{res}(\vec{D}, u')$ and $r' = \max(u, u')$. The unique successor of u' is u'' labelled by $(K, \rho) \geq \vec{D} \mapsto q'$. We show that we can take it as the node corresponding to $(K, \theta\restriction_{r'}) \geq \vec{P} \mapsto q'$. By the invariant in u we know that $\mathit{Comp}(\theta, \mathit{res}(\rho, u))$. Observe that $\mathit{res}(\rho, u'') = \mathit{res}(\rho, u)\restriction_{r'}$ since $\max(u, u') = \max(u, u'')$. Then $\mathit{Comp}(\theta\restriction_{r'}, \mathit{res}(\rho, u''))$ by Lemma 40. Finally, $\mathit{Comp}(\vec{P}, \mathit{res}(\vec{D}, u''))$ thanks to $\vec{T} = \mathit{res}(\vec{D}, u'')$, so the invariant is satisfied.

It is worth noticing that in both cases, the maximal parity seen in $\mathcal{K}(\mathcal{A}, M)$ on the path from u to the new companion node of v' is the same as the one seen on the path between v and v' in $G(\mathcal{A}, M)$. This obvious remark is important so as to establish that when Adam is winning in $\mathcal{K}(\mathcal{A}, M)$, he is also winning in $G(\mathcal{A}, M)$.

C.3 Relating $\mathcal{G}(\mathcal{A})$ and $\mathcal{G}^{st}(\mathcal{A})$

In this subsection we prove

Proposition 24 For every M closed term of type 0: $(M, \emptyset) \geq q$ is winning in $\mathcal{G}(\mathcal{A})$ iff it is winning in $\mathcal{G}^{st}(\mathcal{A})$.

In the proof we essentially show how to find for every stratified residual an "equally good" monotone and stratified residual. While the construction is quite natural, the details are bit heavy. The main part is to show how to transfer a strategy in $\mathcal{G}^{st}(\mathcal{A})$ into the more constrained setting of $\mathcal{G}(\mathcal{A})$. We first treat the easier direction.

Below we will use $\bot_A$ for the least element of $\mathcal{R}^{st}_A$. If we write A in the form $\vec{B} \to o$ then $\bot_{\vec{B}\to o}(\vec{S}) = \emptyset$ for all residuals $\vec{S}$ of the appropriate type.

In order to relate $\mathcal{G}(\mathcal{A})$ and $\mathcal{G}^{st}(\mathcal{A})$ we need to be able to transform elements of $\mathcal{R}_A$ into $\mathcal{R}^{st}_A$ and vice versa. One direction is quite straightforward. Since $\mathcal{R}_o = \mathcal{R}^{st}_o$, for an element $f \in \mathcal{R}_o$ we simply set $f^\diamond = f$. For $f \in \mathcal{R}_{A\to B}$ we put

$$f^\diamond(S) = \begin{cases} (f(g))^\diamond & \text{if } S = g^\diamond \text{ for some } g \in \mathcal{R}_A \\ \bot_B & \text{otherwise.} \end{cases}$$

▶ **Lemma 41.** *For every position $(M,\theta) \geq \vec{f} \mapsto q$ of $\mathcal{G}(\mathcal{A})$: if this position is winning in $\mathcal{G}(\mathcal{A})$ then $(M,\theta^\diamond) \geq \vec{f^\diamond} \mapsto q$ is winning in $\mathcal{G}^{st}(\mathcal{A})$.*

Proof. Given a winning strategy in $\mathcal{G}(\mathcal{A})$ we define a strategy in $\mathcal{G}^{st}(\mathcal{A})$. This strategy just copies the moves in $\mathcal{G}(\mathcal{A})$ staying in positions given by the operation $(\cdot)^\diamond$. As the strategy we obtain this way induces the same sequences of colours, it is obvious that it is winning. ◀

Now we show how to translate a winning strategy of $\mathcal{G}^{st}(\mathcal{A})$ into a winning strategy of $\mathcal{G}(\mathcal{A})$. For this, we first associate to every stratified residual a monotone stratified residual:

$$\begin{aligned} mon(R) =& R \quad \text{for } R \in \mathcal{R}_o^{st} \\ mon(R)(h) =& \bigvee\{mon(R(S)) : mon(S) \leq h\} \\ & \text{for } R \in \mathcal{R}^{st}_{A\to B} \text{ and } h \in \mathcal{R}^{st}_A \end{aligned}$$

Recall that $\mathcal{R}_A$ is a complete lattice for every type A, so the join in the above definition exists. Nevertheless, it requires some calculation to show that $mon(R)$ is indeed in $\mathcal{R}$. Before we can prove the announced transfer of strategies, and hence also Proposition 24, we need to explore the properties of the operation $mon(\cdot)$ and prove that $mon(R)$ is indeed in $\mathcal{R}$.

▶ **Lemma 42.** *For every residual R and every rank $r \in [m]$: $mon(R\restriction_r) = mon(R)\restriction_r$.*

Proof. The lemma is the consequence of the following calculation:

$$\begin{aligned} mon(R\restriction_r)(\vec{h}) =& \bigcup\{R\restriction_r(\vec{S}) : mon(\vec{S}) \leq \vec{h}\} \\ =& \bigcup\{(R(\vec{S}))\restriction_r : mon(\vec{S}) \leq \vec{h}\} \\ =& \big(\bigcup\{R(\vec{S}) : mon(\vec{S}) \leq \vec{h}\}\big)\restriction_r = mon(R)\restriction_r(\vec{h}) \end{aligned}$$

The second equality is direct from the definition of the operation $(\cdot)\restriction_r$. The third from the observation that $(R_1 \cup R_2)\restriction_r = R_1\restriction_r \cup R_2\restriction_r$. ◀

▶ **Lemma 43.** *If $mon(R) \leq f$ then $mon(R\restriction_r) \leq f\restriction_r$.*

Proof. By monotonicity of the operation $(\cdot)\restriction_r$, we get $mon(R)\restriction_r \leq f\restriction_r$. Then we can use Lemma 42. ◀

For the proof of Lemma 46 below we will need one more operation on residuals. Given a rank $r \in [m]$, we write $R^{\geq r}$ for the *truncation* of a residual R to ranks at least r:

$$\begin{aligned} R^{\geq r} =& \{(q,i) \in R : i \geq r\} && \text{if } R \in \mathcal{R}_o \\ R^{\geq r}(S) =& (R(S))^{\geq r} && \text{if } R \in \mathcal{R}_{A\to B} \text{ and } S \in \mathcal{R}_A. \end{aligned}$$

▶ **Lemma 44.** *For every type A, rank $r \in [m]$, and $R \in \mathcal{R}^{st}_A$ we have $R^{\geq r} \in \mathcal{R}^{st}_A$. Moreover, $R\restriction_r = (R^{\geq r})\restriction_r$.*

Proof. In order to verify the (`strat`) property take some $(q,i) \in R^{\geq r}(\vec{S})$, for some $\vec{S}$ of the appropriate type. By definition we have $i \geq r$ and $(q,i) \in R(\vec{S})$. Since R is stratified $(q,i) \in R(\vec{S}\restriction_q)$. So $(q,i) \in R^{\geq r}(\vec{S}\restriction_q)$. The other direction is very similar. The second statement follows by direct examination. ◀

▶ **Lemma 45.** *If $mon(R) \leq f\restriction_r$ then $mon(R^{\geq r}) \leq f$.*

Proof. Take $(q,i) \in mon(R^{\geq r})(\vec{h})$, for some $\vec{h}$ of the appropriate type. We need to show $(q,i) \in f(\vec{h})$. By definition of $mon(\cdot)$, there is $\vec{S}$ such that $mon(\vec{S}) \leq \vec{h}$ and $(q,i) \in R^{\geq r}(\vec{S})$. So $i \geq r$ and $(q,i) \in R(\vec{S})$. The assumption of the lemma gives us $(q,i) \in f\downarrow_r(\vec{h})$. Since $i \geq r$ we obtain $(q,i) \in f(\vec{h})$. ◀

▶ **Lemma 46.** *If $R \in \mathcal{R}^{st}_A$ then $mon(R) \in \mathcal{R}_A$.*

Proof. When A is the base type o, the statement is trivial. So let us consider type $A \to B$ and take some $R \in \mathcal{R}^{st}_{A\to B}$. By definition $mon(R)$ is monotone. It suffices to check the stratification property.

Take $(q,i) \in mon(R)(\vec{h})$. We need to show $(q,i) \in mon(R)(\vec{h}\downarrow_q)$. By definition of $mon(R)$, there is $\vec{S}$ with $mon(\vec{S}) \leq \vec{h}$ and $(q,i) \in R(\vec{S})$. Since R is stratified $(q,i) \in R(\vec{S}\downarrow_q)$. By Lemma 43 we get $mon(\vec{S}\downarrow_q) \leq \vec{h}\downarrow_q$. So $(q,i) \in mon(R)(\vec{h}\downarrow_q)$.

For the opposite direction take $(q,i) \in mon(R)(\vec{h}\downarrow_q)$. We have that $(q,i) \in R(\vec{S})$ for some $\vec{S}$ with $mon(\vec{S}) \leq \vec{h}\downarrow_q$. Lemma 45 gives us $mon(\vec{S}^{\geq rk(q)}) \leq \vec{h}$ and Lemma 44 ensures that $\vec{S}^{\geq rk(q)}$ is a vector of stratified residuals. As R is stratified, $(q,i) \in R(\vec{S}\downarrow_q)$. Once again using Lemma 44 we get $\vec{S}\downarrow_q = (\vec{S}^{\geq rk(q)})\downarrow_q$. So $(q,i) \in R((\vec{S}^{\geq rk(q)})\downarrow_q)$ and once again using stratification of R we obtain $(q,i) \in R(\vec{S}^{\geq rk(q)})$, that gives $(q,i) \in mon(R)(\vec{h})$. ◀

Lemma 25 If $(M,\theta) \geq \vec{S} \mapsto q$ is a winning position for Eve in $\mathcal{G}^{st}(\mathcal{A})$ then for every $\upsilon \geq mon(\theta)$ and every $\vec{f} \geq mon(\vec{S})$, the position $(M,\upsilon) \geq \vec{f} \mapsto q$ is winning in $\mathcal{G}(\mathcal{A})$.

Proof. Given a winning strategy in $\mathcal{G}^{st}(\mathcal{A})$, we show how to play in $\mathcal{G}(\mathcal{A})$ while preserving the invariant given by the lemma. The only complicated case is that of the application rule. The situation in $\mathcal{G}^{st}(\mathcal{A})$ is

$$\begin{array}{c} (NK,\theta) \geq \vec{S} \to q \\ \downarrow \\ (NK,\theta) \geq_R \vec{S} \to q \\ \swarrow \qquad \text{for all } \vec{P} \text{ and } (q',r') \in R(\vec{P}\downarrow_{q'}) \searrow \\ (N,\theta) \geq R\downarrow_q \to \vec{S} \to q' \qquad (K,\theta\downarrow_{r'}) \geq \vec{P}\downarrow_{q'} \to q' \end{array}$$

In this case we should play in $\mathcal{G}(\mathcal{A})$ as follows

$$\begin{array}{c} (NK,\upsilon) \geq \vec{f} \to q \\ \downarrow \\ (NK,\upsilon) \geq_{mon(R)} \vec{f} \to q \\ \swarrow \qquad \text{for all } \vec{h} \text{ and } (q',r') \in mon(R)(\vec{h}\downarrow_{q'}) \searrow \\ (N,\upsilon) \geq mon(R)\downarrow_q \to \vec{f} \to q' \quad (K,\upsilon\downarrow_{r'}) \geq \vec{h}\downarrow_{q'} \to q' \end{array}$$

For the left branches Lemma 42 gives us $mon(R)\downarrow_q = mon(R\downarrow_q)$. So the invariant is satisfied. For the right branches, the condition $(q',r') \in mon(R)(\vec{h})$ means that $(q',r') \in R(\vec{P})$ for some $\vec{P}$ such that $mon(\vec{P}) \leq \vec{h}\downarrow_{q'}$. By monotonicity of $\downarrow_{q'}$ we get $mon(\vec{P})\downarrow_{q'} \leq \vec{h}\downarrow_{q'}$. Using once again Lemma 42 we obtain $mon(\vec{P}\downarrow_{q'}) \leq \vec{h}\downarrow_{q'}$. So the obtained positions satisfy the hypothesis of the lemma, as Lemma 43 guarantees $mon(\theta\downarrow_{r'}) \leq \upsilon\downarrow_{r'}$. ◀

Proposition 24 follows directly from Lemmas 41 and 25.