A Model for Behavioural Properties of Higher-order Programs

Sylvain Salvati[1] and Igor Walukiewicz[2]

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

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.

1998 ACM Subject Classification F.3 Logics and Meaning of Programs

Keywords and phrases Simply typed λY-calculus, Monadic second order logic, semantic models

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.229

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, 5, 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 the `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

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 229–243

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

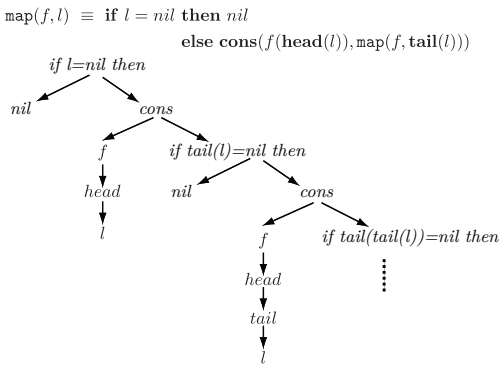

Figure 1 The map function and its semantics in the form of a (simplified) Böhm tree.

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 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 12).

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 [15], where, amongst other contributions, they give a type system to capture the

same dependencies inside terms that we represent in our model. Although the quest for models for behavioural properties has begun some time ago, the results started to appear only recently. Tsukada and Ong [27] extended the approach from [15] to a type 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 at 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 [25]. 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 [25]. Apart from [15], 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]. With respect to other proposals [15, 27, 8] (which capture the so-called Ω-blind automata [22]) our model is capturing automata that are able to detect the divergence symbol Ω: for a run to be accepting, the ranks assigned to the nodes labelled Ω should be even. On a model side, stratification condition is essential.

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 [14, 4, 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. Section 4 shows some consequences of the model construction. The conclusions section outlines some directions for further research. A long version of the paper that gives the details of the proofs is available [24].

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: $\mathit{order}(o) = 0$, and $\mathit{order}(A \to B) = \mathit{max}(1 + \mathit{order}(A), \mathit{order}(B))$. 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* and they respectively have type $(A \to A) \to A$ and A. 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 $\lambda \vec{x}.\ M$. The usual operational semantics of the λ-calculus is given by β-reduction and δ-reduction. The corresponding contraction rules are $(\lambda x.M)N \to_\beta M[N/x]$ and $YM \to_\delta M(YM)$.

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 $t : \mathbb{N}^* \dot{\to} \Sigma \cup \{\Omega\}$ 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 $\delta(q, \Omega^o) = \{\emptyset\}$ if the rank of q is even, and $\delta(q, \Omega^o) = \emptyset$ otherwise[1].

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 (with $k > 0$) 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 labelling 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 *recognizes* 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 appropriate 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 and implicitly assumed that $\mathcal{D}_o$ is of the form $\mathcal{P}(Q)$. It explains the case when we would like to construct a model from an automaton as stated in the theorem below. Let us remark that Theorem 1 allows to make a boolean combination Ω-blind automata when constants have arbitrary interpretation in arbitrary finitary Scott model.

[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.

$$\begin{aligned} [\![x, v]\!] &= v(x) & [\![a, v]\!]h_1 \dots h_k &= \{q : \exists_{(q_1,\dots,q_k)\in\delta(q,a)} q_i \in h_i \text{ for all } i\} \\ [\![\lambda x.M, v]\!]h &= [\![M, v[h/x]]\!] & [\![MN, v]\!] &= [\![M, v]\!]([\![N, v]\!]) \\ [\![Y]\!]\, f &= \bigvee\{f^n(\bot) \mid n \in \mathbb{N}\} & [\![\Omega]\!] &= \bot \end{aligned}$$

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. $[\![M, \emptyset]\!] \in R$. This notion of recognition [23] generalizes the usual notion of recognition for words by finite monoids. In this way, finitary Scott models determine a class of languages of λY-terms they recognize. The following theorem characterizes this class (that Scott domains capture Ω-blind automata was first established in [1]).

▶ **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 of 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 [15] 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.

▶ **Definition 2.** Let $M : o$ be a term of order at most 1. Let v 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* v iff there is a run of $\mathcal{A}$ on $BT(M)$ starting in q, satisfying the conditions of an accepting

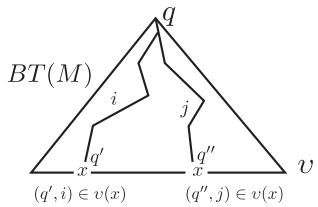

Figure 3 $(q', i), (q'', j) \in \upsilon(x)$ as the maximal color seen on a path from the root to occurrences of x respectively labeled with states q' and q'' are i and j.

run from page 233, 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 handling 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 6 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.

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

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

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 $\upsilon\restriction_r$ means υ where $(.)\restriction_r$ is applied pointwise.

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

$$f_1 \mapsto \dots \mapsto f_k \mapsto q$$

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

▶ **Example 5.** 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 $\delta(q_0, a) = Q \times Q$, $\delta(q_1, b) = Q$, and $\delta(q_0, b) = \delta(q_1, c) = \{\emptyset\}$. 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 $\{(q_1, 1)\} \mapsto (q_0, 0)$. The variable f is meant to represent a closed term so as to make the example not too long. 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 6.** *$[\![M, \upsilon]\!] \geq f_1 \mapsto \dots \mapsto f_k \mapsto q$ 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 are the same nodes) 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

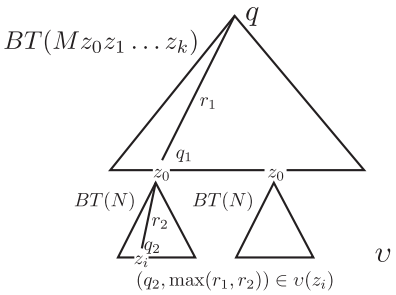

Figure 5 The case of application.

root to z_i this shows that $\mathcal{A}$ accepts az_1z_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 $[\![MN, \upsilon]\!] \geq f_1 \mapsto \cdots \mapsto f_k \mapsto q$, 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 $[\![M, \upsilon]\!]\langle\!\langle N, \upsilon \rangle\!\rangle \geq f_1 \mapsto \cdots \mapsto f_k \mapsto q$, which is the same as $[\![M, \upsilon]\!] \geq \langle\!\langle N, \upsilon \rangle\!\rangle \mapsto f_1 \mapsto \cdots \mapsto f_k \mapsto q$. Now the induction hypothesis tells us that $BT(Mz_0z_1\dots z_k)$ 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 $q' \in [\![N, \upsilon\lfloor_{r'}]\!]$. Using the induction hypothesis for N, we have a run of $\mathcal{A}$ on $BT(N)$ from q' to $\upsilon\lfloor_{r'}$. From these observations we construct a required run on $BT(MNz_1\dots z_k)$ 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 $\upsilon\lfloor_{r_l}$.

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 $(q_2, r_2) \in \upsilon\lfloor_{r_1}(y)$. 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 $\mathcal{R}_o \subseteq \mathcal{P}(Q \times [m])$. 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 $[\![M, v]\!]$ the semantics $[\![Y\,M, v]\!]$ is given by $F_0 \in \mathcal{D}_o$ defined by

$$F_0 = \nu Z_0.\ f^{\partial}(Z_0 \cdot 0 \cup F_1 \cdot 1) \qquad\qquad F_1 = \mu Z_1 \nu Z_0.\ (f\restriction_1)^{\partial}(Z_0 \cdot 0 \cup Z_1 \cdot 1)$$

We omit a, not so short, proof of the correctness of this formula. The proof for the general case is presented in [24]. The set F_0 is the set of states in which the term M is accepted when it is in a context where the maximal color from the root to it is 0 (this includes the empty context), while F_1 is the set of states in which the term M is accepted when the color is 1. This distinction is only important for terms with free variables, where, as we have seen, the values associated to variables by valuations depend on the context. So for closed terms F_0 and F_1 are equal.

3 A model recognizing MSOL properties

We now extend the definitions we have given in the previous section to higher orders. Mellies [16] sketched a definition of fixpoint that only worked for closed terms. We here give a definition of higher-order fixpoints that work for open terms. As ranks in the model are used to keep track of the context where variables occur, most of the technical difficulties of the construction of the model appear in this definition. With this definition, 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 $(\cdot)\restriction_r$, 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:

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

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

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

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 6, in such a run every appearance of $BT(N)$ will be lifted with $\restriction_r$ 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 $\restriction_{rk(q)}$ to g. In other words, for the runs starting in q we should get the same result from $f(g)$ as from $f(g\restriction_{rk(q)})$. Yet another more formal intuition comes from the application clause. The meaning of $\langle\!\langle N, v\rangle\!\rangle$ as a function of v 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 $(\cdot)\lfloor_r$ to higher orders preserves the properties of Lemma 4.

▶ **Lemma 7.** *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 $\mathcal{R}_A$: $(g_1 \vee g_2)\lfloor_r = g_1\lfloor_r \vee g_2\lfloor_r$ and $(g_1 \wedge g_2)\lfloor_r = g_1\lfloor_r \wedge g_2\lfloor_r$.

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 $(\cdot)\lfloor_r$; 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:

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

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

▶ **Lemma 8.** *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 $g^\partial \in \mathcal{D}_A$.*

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:

$$[\![Y^A, v]\!]h = \mathrm{fix}(h, 0) = \nu f_0.h^\partial\Big(f_0 \cdot 0 \vee \bigvee_{i=1}^{m} \mathrm{fix}(h, i) \cdot i\Big) \ .$$

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

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

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:

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

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 8 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 9.** *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 10.** *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 $=_{\beta\delta\eta}$, or, put differently, that we have constructed a model of λY-calculus.

▶ **Proposition 11.** *For every M, N and υ, if $M =_{\beta\delta\eta} N$, 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 12** (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 13.** Continuing the example from page 236 we will calculate the value of the term $M^{o\to o} = Y(\lambda f x.a\, x\, (f(b\, x)))$. This term is a simplified version of the `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 $[\![\lambda f x.a\, x(f(b\, x))]\!] \geq g$ 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 $[\![M, \emptyset]\!] \geq h$ which finally gives us $q_0 \in [\![Mc]\!]$.

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 $\mathit{Terms}(\Sigma, \mathcal{T}, X)$ for the set of terms over a signature Σ, such that all their (free or bound) variables are from $\mathcal{X}$, and all their subterms have types in $\mathcal{T}$.

▶ **Theorem 14** ([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 $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ 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 12. The model $\mathcal{D}^{\mathcal{A}}$ is finite in every type. So the set of possible semantical values of terms from $\mathit{Term}(\Sigma, \mathcal{T}, \mathcal{X})$ 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 [3]. 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* [2]. 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 $[\![N_2]\!]$. 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. [26]). 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.

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 (be it as combinations of least and greatest fixpoints, or other kinds of fixpoints) interpretations of fixpoints is unknown. In particular, we may wonder whether they capture properties beyond those expressible with parity automata.

References

1 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(1):1–23, 2007.

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

3 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.

4 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.

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

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 Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. In *FOSSACS 15*, volume 9034 of *LNCS*, pages 41–55, 2015.

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

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

16 P.A. Mellies. Linear logic and higher-order model checking, June 2014. http://www.pps.univ-paris-diderot.fr/~mellies/slides/workshop-IHP-model-checking.pdf.

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 Sylvain Salvati. Recognizability in the Simply Typed Lambda-Calculus. In *WOLIC*, volume 5514 of *LNCS*, pages 48–60, 2009.

24 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs, April 2015. https://hal.archives-ouvertes.fr/hal-01145494.

25 Sylvain Salvati and Igor Walukiewicz. Typing weak MSOL properties. In *FOSSACS 15*, volume 9034, pages 343–357, 2015.

26 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.

27 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.