LambdaY-Calculus With Priorities

Igor Walukiewicz
CNRS, Bordeaux University

Abstract—**The lambdaY-calculus with priorities is a variant of the simply-typed lambda calculus designed for higher-order model-checking. The higher-order model-checking problem asks if a given parity tree automaton accepts the Böhm tree of a given term of the simply-typed lambda calculus with recursion. We show that this problem can be reduced to the same question but for terms of lambdaY-calculus with priorities and visibly parity automata; a subclass of parity automata. The latter question can be answered by evaluating terms in a simple powerset model with least and greatest fixpoints. We prove that the recognizing power of powerset models and visibly parity automata are the same. So, up to conversion to the lambdaY-calculus with priorities, powerset models with least and greatest fixpoints are indeed the right semantic framework for the model-checking problem. The reduction to lambdaY-calculus with priorities is also efficient algorithmically: it gives an algorithm of the same complexity as direct approaches to the higher-order model-checking problem. This indicates that the task of calculating the value of a term in a powerset model is a central algorithmic problem for higher-order model-checking.**

I. Introduction

Higher-order model-checking has become a successful foundation for verification of higher-order programs. At first restricted to call-by-name purely functional programs, in recent years its scope has been substantially enlarged [1]–[5].

Technically, the model-checking problem can be stated as follows: given a term of a simply typed λ-calculus with fixpoints, and a parity tree automaton, decide if the Böhm tree of the term is accepted by the automaton. The Böhm tree of the term is a generalization of the notion of the result of a computation to potentially non-terminating computations. Decidability of the higher-order model-checking problem was proved by Ong [6]. Since then it has been has been reproved using several different methods [7]–[12]. Among them, a model-based approach is the most relevant for this paper.

The idea of the model-based approach is to construct a finite model *recognizing* a given property [13]. A model recognizes a property if the value of a term in the model determines if the Böhm tree of the term satisfies the property. This is analogous to a fundamental concept of recognizability by semigroups in formal language theory. The model-based approach allows to deduce in an elegant way many results about higher-order model checking [14], [15]. Unfortunately, the model constructions we know of are quite complicated. More seriously, it is not clear what is a suitable class of models that plays the same role as semigroups in the case of languages of finite words. It is even not known what kinds of fixpoints are need to construct models recognizing properties given by parity automata.

In this paper we show that the simplest possible class of models, namely that of models based on a finite powerset lattice and monotone functions with least and greatest fixpoints, corresponds exactly to, a certain refinement of, the higher-order model-checking problem. The refinement consist of a finer typing system that we call λY-calculus with priorities, and a restriction of parity automata to what we call visibly parity automata.

Our result extends the one for automata with trivial acceptance conditions[1]. Aehlig [16] has shown that properties defined by such automata can be recognized by powerset models with recursion interpreted as the greatest fixpoint. Such models are also called Scott models in the literature, although most often they are considered over arbitrary directed complete partial-orders, and not necessarily finite distributive lattices. Actually, recognizing power of automata with trivial acceptance conditions, and finitary powerset models with greatest fixpoint interpretation is the same [15]. Thus to go beyond automata with trivial acceptance conditions we need to enlarge the class of interpretations.

Since complete lattices have both least and greatest fixpoints, it is tempting to use both in the semantics. As we have only one recursion operator in the calculus, it is not clear which fixpoint to use where. Observe that using just least fixpoints would give dual models, and would not give more recognizing power than using just greatest fixpoints.

In this paper we propose the λY*-calculus* with priorities, a calculus where every recursion operator, and every constant is indexed with a priority. Recursion operators with even priorities are interpreted as the greatest fix points, and those with odd priorities as the least fix points. The main point is to relate this semantics to acceptance by automata. Having constants indexed by priorities leads to a notion of *visibly parity automata* where the priorities are not associated to states but to letters read by the automaton. Our main result, Theorem 14, states that there is a perfect match between models and automata: recognizing power of powerset models under such interpretation is equivalent to that of visibly parity automata.

Extending the comonadic translation of Melliès [17], we show that for every assignment of priorities to constants: every term of the λY-calculus can be translated to a term of the λY-calculus with priorities such that the two terms have the same Böhm trees. This allows to reduce the higher-order model-checking problem to the model-checking problem

978-1-7281-3608-0/19/\$31.00 ©2019 European Union

[1]All automata in this paper are $\perp$-blind; called Ω-blind in [15]. We discuss this restriction in the main text and in the conclusions.

for λY-calculus with priorities and visibly parity automata. In consequence, the higher-order model-checking problem can be solved by evaluation in simple power-set models. Moreover, this reduction can be done in polynomial time, and the resulting algorithm has the same complexity as other known approaches [9], [18] . This confirms the central position of the algorithmic problem of evaluating terms with least and greatest fix points in the powerset model.

To sum up, the main technical contributions of the paper are the following:

- Definition of the λY-calculus with priorities.
- Characterization of its semantics in powerset models in terms of acceptance by visibly parity automata.
- Extension of the co-monadic translation of Melliès to terms with fixpoints.

In this paper we propose a framework for higher-order model-checking with a very simple semantic interpretation. We hope that this is a step towards Eilenberg-like variety theory for λY-calculus. The model-based approach puts a focus on computing fixpoints in finite lattices. The model-checking of the propositional mu-calculus is the most known instance of this problem, but the higher-order version is no less intriguing.

Related work: This work relies on some important insights to higher-order model-checking. An idea of tracking priorities in a type system was introduced in a seminal paper of Kobayashi and Ong [8]. The comonadic nature of priorities and the translation on terms proposed by Melliès [17] is another cornerstone of this work. The paper of Kobayashi, Lozes and Bruse [19] was the starting inspiration for this work; it implies that Melliès' translation leads to a reduction of higher-order model-checking to evaluation in powerset models. The present paper belongs to the line of research on models for higher-order model-checking. Apart from the work of Aehlig mentioned above, we can mention approaches of Tsukada and Ong [10], as well as Grellois and Melliès [11], [20]. In both works the fixpoint operator is defined via a parity game and is somehow external to a model. Even closer are the works of Salvati and Walukiewicz culminating in a model construction for all ω-regular properties [12]. All these works use models enriched with priorities, inspired by intersection types of Kobayashi and Ong. In the present paper, priorities are in the syntax, and not in the model. This changes many things, but there are also many techniques that can be reused. Bruse [21] considers Krivine machine interpretation for higher-order fixpoint logic, so he needs to deal with both higher-order and both types of fixpoints. The acceptance condition for his machines reduces to the parity condition for terms typable in our system. A recent paper of Melliès [22] introduces a notion of higher-order parity automata. Their behavior is somehow similar to our semantic games (game PSG on page 11). The objectives of op. cit. are quite different from ours, and so are techniques except of Melliès' translation. In a broader context, this paper is a part of continuing effort to understand better the higher-order model-checking problem [23]–[25].

Structure of the paper: In the next section we recall basic notions behind the higher-order model-checking problem. We describe the correspondence between automata with trivial acceptance conditions, and powerset models with greatest fixpoint interpretation. Section III introduces λY-calculus with priorities, and visibly parity automata. It explains how to reduce the model-checking problem to that for visibly parity automata. Section IV presents main results of the paper. It also states the main technical theorem whose proof is outlined in Section V. Section VI shows how to translate λY-terms to λY-terms with priorities. Section VII discusses applicability of the results to algorithmics of higher-order model-checking. All missing proofs are included in the full version of the paper [26].

Acknowledgments: The author sincerely thanks the reviewers for their careful reading of the paper and their useful comments.

II. The λY-calculus and parity automata

In this section we recall definitions of the λY-calculus, and of parity automata. We also recall the characterization of the recognizing power of parity automata with trivial acceptance conditions in terms of simple models of the λY-calculus where fixpoint operators are interpreted as greatest fixpoints.

λY-calculus: The λY-calculus is simply-typed lambda calculus with a fixpoint operator. The set of simple types is constructed from a unique base type o using a binary operation $\to$. As usual we shall write $A_1 \to \cdots \to A_k \to B$ for $(A_1 \to (\dots(A_k \to B)\dots))$. We use *Types* for the set of all simple types.

An *alphabet* is a set Σ of typed constants. Every constant $b \in \Sigma$ has an arity $ar(b)$ that is a strictly positive natural number. A constant b of arity $ar(b)$ has a type

$$b : o \to \cdots \to o \to o,$$

where there are $ar(b)$ arrows. We only allow this shape of types for constants. This is a standard restriction in the context of higher-order model-checking, except maybe for allowing constants of the base type o. We disallow constants of type o for notational convenience.

Terms of the λY-calculus are built from variables and constants in Σ with the help of abstraction, application, and fixpoint operations. We use $x, y, \dots$ and F with subscripts for variables. We assume that variables are typed but we will seldom write their type explicitly. Construction of terms is subject to the standard type discipline. If M is a term of type B and x a variable of type A, then $\lambda x.M$ is a term of type $A \to B$. If M is a term of type $A \to B$ and N is a term of type A then $M \cdot N$ is a term of type B. We will often write MN instead of $M \cdot N$. Finally, if M is a term of type A, and F is a variable of type A then $YF.M$ is a term of type A. So we adopt a syntax where Y is a binder, and not a fixpoint combinator.

The usual operational semantics of the calculus is given by β and δ-reductions (we omit the standard definition of a substitution): $(\lambda x.M) \cdot N \to_\beta M[N/x]$, and $YF.M \to_\delta$

$M[(YF.M)/F]$. We write $\to^*_{\beta\delta}$ for reflexive and transitive closure of the union of the two relations.

Böhm trees of terms: Böhm tress are a kind of normal forms for λY-terms. They may be infinite, since the calculus does not have a strong normalization property.

Let us fix an alphabet Σ as above. Let $\perp$ be a special symbol not in Σ. We write $\Sigma_\perp$ for $(\Sigma \cup \{\perp\})$. A, potentially infinite, $\Sigma_\perp$-tree is a partial function $t : (\mathbb{N}_{>0})^* \stackrel{\cdot}{\to} \Sigma_\perp$. For a node $v \in (\mathbb{N}_{>0})^*$ and a direction $i \in \mathbb{N}_{>0}$ we call vi the i-th successor of v. This successor may not exist if $t(vi)$ is not defined. We require that for every node $v \in (\mathbb{N}_{>0})^*$, if the constant $b = t(v)$ has an arity $k = ar(b)$ then v has k successors $v1, \dots, vk$, and has no other successors. If $t(v) = \perp$ then v should have no successors.

Definition 1 (Böhm tree) A *Böhm tree* of a closed term M of type o, denoted $BT(M)$, is a $\Sigma_\perp$-tree defined recursively:

- if $M \to^*_{\beta\delta} bN_1 \dots N_{ar(b)}$ for some constant $b \in \Sigma$ then $BT(M)$ has the root labeled b with subtrees of the root being $BT(N_1), \dots, BT(N_{ar(b)})$;
- otherwise $BT(M) = \perp$.

Thanks to subject reduction and confluence of $\to^*_{\beta\delta}$, every term has a unique Böhm tree [27]. Because of our assumption on the shape of type of constants in Σ, all terms N_i in the first clause of the definition must be closed and of type o. For the same reason, all leaves in $BT(M)$ must be labeled with $\perp$. In what follows it is possible to add constants of type o without problems. Constants of higher-order types, like $(o \to o) \to o$, would introduce variables and bindings in Böhm trees. In consequence, it would not be clear how to run a tree automaton on such Böhm trees.

Alternating parity automata: We use alternating (max)parity automata to express properties of Böhm trees. The definition is standard except for the case when an automaton reaches a leaf labeled $\perp$: it accepts no matter what state it is in. We will discuss this phenomenon below.

A *parity automaton* is a tuple

$$\mathcal{A} = \langle Q, \Sigma, \{\delta_b\}_{b\in\Sigma}, \Omega : Q \to \{0, \dots, p\}\rangle\ ,$$

where Q is a finite set of states, Σ is an alphabet,

$$\delta_b : Q \to \{(S_1, \dots, S_{ar(b)}) : S_i \in \mathcal{P}(Q), i = 1, \dots, ar(b)\}$$

is a transition function, and Ω is an assignment of priorities to states. Priorities are integers between 0 and p. As before, we assume that every $b \in \Sigma$ has its arity $ar(b)$. For readability, we will write $\delta(q, b)$ for $\delta_b(q)$.

Parity automata run on $\Sigma_\perp$-trees. An *acceptance game* for $\mathcal{A}$ from $q \in Q$ on a $\Sigma_\perp$-tree $t : (\mathbb{N}_{>0})^* \stackrel{\cdot}{\to} \Sigma_\perp$ involves two players called Adam and Eve. Eve starts in (q, ε) namely in the state q and in the root node of t. She looks at the letter $b = t(\varepsilon)$ in the root. If $b = \perp$ then Eve wins, otherwise Eve needs to choose some $(S_1, \dots, S_{ar(b)}) \in \delta(q, b)$. Next, Adam chooses i_1 and $q_{i_1} \in S_{i_1}$. The game proceeds to position (q_{i_1}, i_1), and a new turn starts. If a player cannot make a move, she looses; for example Eve looses if $\delta(q, b) = \emptyset$, and Adam looses if Eve can choose $(\emptyset, \dots, \emptyset)$. The winner of an infinite play is decided by looking at the sequence of states $q_{i_1}, q_{i_1 i_2}, \dots$ encountered during the play. Eve wins if the maximal priority of a state seen infinitely often is even.

Automaton $\mathcal{A}$ *accepts a tree t from q* if Eve has a winning strategy in the game described above from (q, ε) on t. Over infinite trees without $\perp$ the power of our parity automata is the same as that of monadic second-order logic. Our automata are *$\perp$-blind*, meaning that they accept when they reach a leaf labeled $\perp$. (In [15] this property is called Ω-blind, but here we use $\perp$ to denote divergence). For example, the language "there is a leaf labeled $\perp$" is not recognized by our automata. This strange behavior is quite common in the literature on higher-order model checking [1]. As we will see in the next subsection, it is a consequence of the way divergence is handled in models of the simply typed lambda-calculus.

We finish this subsection with a upper closure operation on automata.

Definition 2 ($\mathrm{up}(\mathcal{A})$) For a transition function δ_b, its upper closure $\mathrm{up}(\delta_b)$ is defined by: $(S_1, \dots, S_k) \in \mathrm{up}(\delta_b)(q)$ if there is $(S'_1, \dots, S'_k) \in \delta_b(q)$ with $S'_i \subseteq S_i$, for $i = 1, \dots, k$. Automaton $\mathrm{up}(\mathcal{A})$ is $\mathcal{A}$ with transition functions changed from $\{\delta_b\}_{b\in\Sigma}$ to $\{\mathrm{up}(\delta_b)\}_{b\in\Sigma}$.

From the definition of acceptance it should be clear that a tree is accepted from a state q by $\mathrm{up}(\mathcal{A})$ iff it is accepted from q by $\mathcal{A}$.

GFP*-semantics and automata with trivial acceptance conditions:* In this last part of the introductory section we recall a close relation between automata with trivial acceptance conditions, and simple models of λY-calculus where fixpoint operators are interpreted as greatest fixpoints (GFP).

Definition 3 (Finitary powerset model) A *finitary powerset model* of a signature Σ is a tuple $\mathcal{D} = \langle \{D_A\}_{A\in Types}, \{[\![b]\!]^{\mathcal{D}}\}_{b\in\Sigma}\rangle$, where D_o is the lattice $\mathcal{P}(Q)$ for some set Q, and for every type $A \to B$, lattice $D_{A\to B}$ is the set of monotone functions from D_A to D_B ordered coordinate-wise. An interpretation $[\![b]\!]^{\mathcal{D}}$ of a constant $b \in \Sigma$ of a type B is an element of D_B.

We need a lattice structure in the model to interpret fixpoint operators. Later, when we will consider complexity of some decision problems, it will be important that the lattice is distributive. As every finite distributive lattice is isomorphic to a lattice of sets, we prefer for simplicity to start with a powerset lattice immediately.

The GFP-*semantics* of terms in such a model is standard, but for the fact that all fixpoints are interpreted as the greatest fixpoints. Since every D_A is a finite lattice, every monotone function in $D_{A\to A}$ has the least and the greatest fixpoint, denoted LFP, and GFP respectively. For now we will use only the greatest fixpoints. We will use both types of fixpoints to interpret λY-calculus with priorities.

We spell out the definition of the semantics of a λY-term M in a valuation ϑ and a model $\mathcal{D}$, in symbols $[\![M, \vartheta]\!]^{\mathcal{D}}_{\mathsf{GFP}}$. We keep the subscript GFP to remind that we use only greatest fixpoints. On the other hand, we will often omit the superscript $\mathcal{D}$ for readability. As usual, a valuation is a function assigning to every variable of type A a value from D_A. The definition of $[\![M,\vartheta]\!]^{\mathcal{D}}_{\mathsf{GFP}}$ is by induction on the size of M.

- $[\![x,\vartheta]\!]_{\mathsf{GFP}} = \vartheta(x)$,
- $[\![b,\vartheta]\!]_{\mathsf{GFP}} = [\![b]\!]^{\mathcal{D}}$,
- $[\![\lambda x.M,\vartheta]\!]_{\mathsf{GFP}} = \boldsymbol{\lambda} h.[\![M,\vartheta[h/x]]\!]_{\mathsf{GFP}}$,
- $[\![MN,\vartheta]\!]_{\mathsf{GFP}} = [\![M,\vartheta]\!]_{\mathsf{GFP}}([\![N,\vartheta]\!]_{\mathsf{GFP}})$,
- $[\![YF.N,\vartheta]\!]_{\mathsf{GFP}} = \mathsf{GFP}\,\boldsymbol{\lambda} h.[\![N,\vartheta[h/F]]\!]_{\mathsf{GFP}}$.

It is well-known that the interpretation of a term is always a monotone function, and that this interpretation is sound with respect to β and δ reductions [27].

Models can be constructed from automata as follows.

Definition 4 (Model $\mathcal{D}^{\mathcal{A}}$) For an automaton $\mathcal{A} = \langle Q,\Sigma,\{\delta_b\}_{b\in\Sigma},\Omega\rangle$ the model $\mathcal{D}^{\mathcal{A}}$ has $\mathcal{P}(Q)$ as the interpretation of the base type; a constant b is interpreted as

$$[\![b]\!]_{\mathsf{GFP}}(S_1,\dots,S_{ar(b)}) = \{q : (S_1,\dots,S_{ar(b)}) \in \mathrm{up}(\delta_b(q))\}\ .$$

Automata can be constructed from models.

Definition 5 (Automaton $\mathcal{A}^0_{\mathcal{D}}$) For a finitary powerset model $\mathcal{D}$ over the base set $\mathcal{P}(Q)$ we define a parity automaton $\mathcal{A}^0_{\mathcal{D}} = \langle Q,\Sigma,\{\delta_b\}_{b\in\Sigma},\Omega : Q\to\{0\}\rangle$ where

$$\delta_b(q) = \{(S_1,\dots,S_{ar(b)}) : q \in [\![b]\!]_{\mathsf{GFP}}(S_1,\dots,S_{ar(b)})\}$$

There is no way to read an assignment of priorities Ω from the model. So in the above definition we just take the trivial one. This choice is justified by Proposition 7 below.

The class of automata we obtain by this construction is important enough to give it a name. We say that an automaton has a *trivial acceptance condition* if all the states have priority 0, i.e., $\Omega(q) = 0$ for all states q. We will write $\mathcal{A}^0$ when we want to stress that $\mathcal{A}$ has a trivial acceptance condition.

Fact 6 Fix an alphabet Σ. For every parity automaton with trivial acceptance condition $\mathcal{A}^0$ over Σ, and every finitary powerset model $\mathcal{D}$ over Σ:

$$\mathcal{A}^0_{\mathcal{D}^{\mathcal{A}^0}} \text{ is } \mathrm{up}(\mathcal{A}^0), \quad \text{and } \mathcal{D}^{\mathcal{A}^0_{\mathcal{D}}} \text{ is } \mathcal{D}.$$

This fact is one of the reasons why we have restricted to powerset models. The constructions can be quite easily extended to arbitrary finite lattice models, but the equivalence from the above fact becomes less direct.

A *model $\mathcal{D}$ can recognize a set of closed terms of type o*: the set of terms recognized by a set $F \subseteq D_o$ is

$$\{M : [\![M]\!]^{\mathcal{D}}_{\mathsf{GFP}} \in F,\ M \text{ closed term of type } o\}\ .$$

An *automaton $\mathcal{A}$ also can recognize a set of closed terms of type o*: we can choose a state q and consider those terms whose Böhm trees are accepted by $\mathcal{A}$ from q.

The main point of the correspondence from Fact 6 is that an automaton and its corresponding model recognize the same sets of terms. (Recall that $\mathcal{A}$ and $\mathrm{up}(\mathcal{A})$ recognize the same sets of terms.) The proposition below is a reformulation of results form [15], [16].

Proposition 7 Fix an alphabet Σ. Let $\mathcal{A}^0$ be an automaton with a trivial acceptance condition over the alphabet Σ, and let $\mathcal{D}^{\mathcal{A}^0}$ be the corresponding powerset model. For every closed λY-term M of type o over the signature Σ:

$$[\![M]\!]^{\mathcal{D}^{\mathcal{A}^0}}_{\mathsf{GFP}} = \{q : \mathcal{A}^0 \text{ accepts } BT(M) \text{ from } q\}\ .$$

Due to Fact 6, the same equality holds when we start with a model $\mathcal{D}$ and consider the automaton $\mathcal{A}^0_{\mathcal{D}}$:

$$[\![M]\!]^{\mathcal{D}}_{\mathsf{GFP}} = \{q : \mathcal{A}^0_{\mathcal{D}} \text{ accepts } BT(M) \text{ from } q\}\ .$$

This shows that the recognizing power of finitary powerset models with GFP-interpretation is the same as that of automata with a trivial acceptance condition.

III. The λY-calculus with priorities

Proposition 7 puts a limit on what can be recognized with finitary powerset models using only greatest fixpoints. But we have also least fixpoints available in powerset models, so one may ask what is the recognizing power of finitary powerset models when we use both types of fixpoints. To give an answer to this question, we propose a syntax allowing to indicate when Y should be interpreted as the least and when as the greatest fixpoint. The challenge is to do it in a way that still preserves a relation to acceptance by automata.

The *λY-calculus with priorities* results by adding priorities to the syntax. Priorities appear as superscripts over applications and over fixpoint binders. The simple type discipline of the λY-calculus is also refined to priority types.

Priority types are simple types annotated with priorities:

$$\theta = o \mid \tau \to \theta \qquad \text{where} \qquad \tau = (r,\theta) \qquad r \in \mathbb{N}$$

There is only one base type o. Only types to the left of an arrow have a priority annotation, while the base type is not annotated. To every priority type θ naturally corresponds a simple type A_θ obtained by hereditary erasing priority annotations.

Priority types are Kobayashi and Ong types [8] without conjunction. As we will see later, we avoid the conjunction thanks to an extended Melliès translation from Section VI and two kinds of typing assertions, $(=,\tau)$ and $(\le,\tau)$, in typing environments.

Terms are built from variables and constants, using abstraction, priority application, and priority fixpoint operator. In particular, $N \cdot_r K$ is a term when N and K are terms, and r is a priority. Similarly, $Y^r F.N$ is a term when r is a priority, F is a variable, and N is a term. The rest of the constructs are standard: a variable, x or F, is a term; a constant b is a term; and an abstraction $\lambda x.N$ is a term, if N is a term. We use two kinds of symbols for variables, $x, y, \dots$ for those bound by λ, and F for those bound by Y. There are no priorities on λ-abstractions.

As for λY-calculus, constants are typed. We write Σ^{pr} for a set of constants with priorities: constant $b \in \Sigma^{pr}$ has not only its arity, $ar(b)$, but also its priority $pr(b)$. The type of a constant b of arity $k = ar(b)$ and priority $r = pr(b)$ is

$$b : (r,o) \to \dots \to (r,o) \to o,$$

where there are k arrows. The fact that all arguments have the same priority is not important, it is done only for notational convenience.

Terms are subject to a typing discipline presented in Figure 1. It is a refinement of simple types, in a sense that every typable term is typable in simple types obtained by erasing the priority annotation. We still write judgments as $\Gamma \vdash M : \theta$, hoping that types and terms indicate when we mean typing with priority types, and when typing with simple types. Environments appearing to the left of typing judgments are functions from variables to assumptions of the form $(=,\tau)$ or $(\leq,\tau)$, where τ is a pair (r,θ) with r a priority and θ a priority type. We will write environments as lists, for example: $x = (2,o), y \leq (1,(3,o) \to o)$. Observe that $x = (2,o), x \leq (3,o)$ is not an environment, as x has two priority types. The operation $\Gamma\restriction_r$ used in the application rule

$$\Gamma \vdash b : \theta \qquad \theta \text{ is the type of } b$$

$$\Gamma, x = (0,\theta) \vdash x : \theta \qquad \Gamma, x \leq (r,\theta) \vdash x : \theta$$

$$\frac{\Gamma, x = (r,\theta_1) \vdash M : \theta_2}{\Gamma \vdash \lambda x.M : (r,\theta_1) \to \theta_2}$$

$$\frac{\Gamma \vdash M : (r,\theta_1) \to \theta_2 \qquad \Gamma\restriction_r \vdash N : \theta_1}{\Gamma \vdash M \cdot_r N : \theta_2}$$

$$\frac{\Gamma, F = (r,\theta) \vdash N : \theta}{\Gamma, \Delta \vdash Y^r F.N : \theta} \qquad \begin{array}{c}\text{all assumptions in } \Gamma\\ \text{have priorities } \geq r\end{array}$$

Fig. 1. Typing rules of λ-calculus with priorities.

is defined by: for all x and θ,

- change $x = (r,\theta)$ in Γ to $x \leq (r,\theta)$; and
- remove $x = (i,\theta)$ and $x \leq (i,\theta)$, for all $i < r$.

Example: Consider a constant b of arity 2 and priority 3. Let Γ be the environment $x \leq (6,o), y = (3,o)$. We have a typing

$$\frac{\dfrac{\Gamma \vdash b : (3,o) \to (3,o) \to o \quad \Gamma\restriction_3 \vdash x : o}{\Gamma \vdash b \cdot_3 x : (3,o) \to o} \qquad \Gamma\restriction_3 \vdash y : o}{\Gamma \vdash (b \cdot_3 x) \cdot_3 y : o}$$

where $\Gamma\restriction_3$ is $x \leq (6,o), y \leq (3,o)$. Observe that we do not get a typing for Γ' of the form $x \leq (6,o), y = (2,o)$. This is because $\Gamma'\restriction_3$ does not have an assumption on y. Similarly, if we took Γ'' with $y = (5,o)$ instead then $\Gamma''\restriction_3$ would have $y = (5,o)$ and derivation $\Gamma''\restriction_3 \vdash y : o$ would be impossible.

Observation: If every constant has priority 0, namely its type is of the form $(0,o) \to \dots \to (0,o) \to o$ then all typing rules can use only applications and fixpoints of priority 0: $N \cdot^0 K$ and $Y^0 F.N$. In this case the typing rules become just the typing rules of the λY-calculus as all typing environments will use only priority 0. The picture is more complicated if every constant has priority 1. Indeed, to type the term $\lambda x.x$ we need priority 0, as its types have the form $(0,\theta) \to \theta$.

The computation rules of the λY-calculus with priorities are β and δ-reductions. As expected they preserve priority typing.

Lemma 8 (Subject reduction for priority typing) If $\Gamma \vdash (\lambda x.M) \cdot^r N : \theta$ then $\Gamma \vdash M[N/x] : \theta$. If $\Gamma \vdash Y^r F.M : \theta$ then $\Gamma \vdash M[Y^r F.M/F] : \theta$.

We define the Böhm tree of a priority term M, $BT(M)$, in the same way as we have done for λY-terms (Definition 1). To a priority term M corresponds a λY-term $\overline{M}$ obtained by removing priorities in applications and fixpoint operators. It is easy to verify that $\overline{M}$ is simply typable and that $BT(M) = BT(\overline{M})$.

Semantics: The first gain from introducing priorities in the syntax is that we can now refine the semantics of terms. We evaluate priority λ-terms in finitary lattice models as in Definition 3. The difference with GFP-interpretation is that now we use both the least and the greatest-fixpoints. Recall that to every priority type θ corresponds a simple type A_θ obtained by hereditary removing priorities in θ. The meaning of a term of type θ is an element of D_{A_θ}. The definition of the semantics is verbatim the same as for GFP-interpretation, but for the meaning of fixpoints:

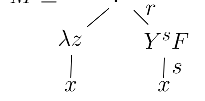
$$\begin{array}{l}[\![Y^r F.N, \vartheta]\!] = \mathsf{LFP}\,\boldsymbol{\lambda} h.[\![N, \vartheta[h/F]]\!] \quad \text{if } r \text{ is odd, and}\\ \qquad\qquad\qquad\quad \mathsf{GFP} \quad \text{instead of } \mathsf{LFP} \text{ if } r \text{ is even.}\end{array}$$

Observe that priorities do not influence the meaning of the application.

A. Terms with priorities are priority-homogeneous

The main point about terms with priorities is that for every variable, all its occurrences have "the same application priority". This is the crucial property that is behind all the results presented in this paper.

Figure 2 gives an example of how to think about application priorities. Consider a tree representation of a term with λx and $Y^s F$ having one successor, and the application $\cdot_r$ symbol having two successors. The right edge of $\cdot_r$ has priority r. The edge from Y^s has priority s. The left edge of the application, and all other edges have label 0. In this representation, the application priority between two positions is the maximum priority on the edges of the path between the positions. A formal definition is given below.



$apr(x, M) = \{0, r \oplus s\}$

Fig. 2. Application rank of variable x in term $M \equiv (\lambda z.x) \cdot^r (Y^s F.x)$.

Definition 9 We define the *set of application priorities of variable in a term*, $apr(x, M)$, by induction on the structure of M. Below, $r \oplus s$ stands for the priority $\max(r, s)$, and $r \oplus S$ stands for the set $\{r \oplus s : s \in S\}$.

- $apr(x, M) = \emptyset$ if x is not free in M;
- $apr(x, x) = \{0\}$;
- $apr(x, \lambda z.N) = apr(x, N)$ if $x \neq z$;
- $apr(x, Y^r F.N) = r \oplus apr(x, N)$ if $x \neq F$;
- $apr(x, N \cdot_r K) = apr(x, N) \cup (r \oplus apr(x, K))$

Definition 10 A term M is *priority-homogeneous* if

- for every subterm of the form $\lambda x.N$, the set $apr(x, N)$ is a singleton or the empty set.
- for every subterm $Y^r F.N$, we have $apr(F, N) = \{r\}$ or $apr(F, N) = \emptyset$.

The next lemma says that all priority typable terms are priority-homogeneous. The opposite direction is not true because of the fixpoint rule.

Lemma 11 If $\Gamma \vdash M : \theta$ then M is priority-homogeneous and the following properties hold:

- if $x = (r, \theta_x)$ is in Γ then $apr(x, M) = \{r\}$ or $apr(x, M) = \emptyset$ (in the latter case, x does not appear in M).
- if $x \leq (r, \theta_x)$ is in Γ then $\max(apr(x, M)) \leq r$, or $apr(x, M) = \emptyset$.

Since every priority typable term is priority homogeneous, we can also put a priority next to λx the same way as we do with the fixpoint $Y^r F$. We could also remove r superscript from Y. Yet we prefer the present, slightly asymmetric, syntax since we will need priorities for Y to define the semantics, but priorities on λ will not be useful.

Example: Not all priority-homogeneous terms are priority typable. The term $Y^r F.x \cdot_r F$ is priority-homogeneous. This term would be priority-typable if there were no restriction on Γ in the fixpoint rule, but it is not typable with this restriction. The unfolding of this fixpoint term is $x \cdot_r (Y^r F.x \cdot_r F)$. It is not priority-homogeneous. In this term the application priority of the first occurrence of x is 0 while the second occurrence has application priority r.

B. Visibly parity automata, and their recognizing power

In Σ^{pr} every constant $b \in \Sigma$ has its priority $pr(b)$. It makes sense to consider parity automata whose priority function depends on letters and not on states.

A *visibly parity automaton* is

$$\mathcal{A} = \langle Q, \Sigma^{pr}, \{\delta_b\}_{b \in \Sigma^{pr}}, pr : \Sigma^{pr} \to \{0, \dots, p\}\rangle$$

where pr is the priority function coming with Σ^{pr}. The notion of accepting a tree from a state is the same as before for parity automata, but pr is used instead of Ω. This means that the priority depends on a letter read and not on the current state.

Of course, visibly parity automata are weaker than parity automata. For example, they cannot express a property "there is a path on which b appears infinitely often". Visibly parity automata look rather trivial from the point of view of automata theory. Yet, they are sufficient for model-checking of transition systems, via the translation we explain below. They also offer a potential advantage because elimination of alternation and Boolean operations are much easier for visibly parity automata than for parity automata.

We argue that in the context of recognizing Böhm trees of terms, visibly parity automata are sufficiently expressive. Indeed, once a maximal priority p is fixed, there is an operation on trees and automata such that $exp_p(t)$ is accepted by $exp_p(\mathcal{A})$ iff t is accepted by $\mathcal{A}$ (cf. Figure 3). Moreover, this operation is easy to implement on terms.

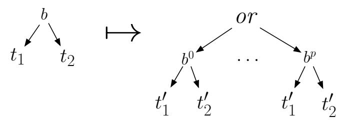


Fig. 3. The tree expansion operation, $exp_p(t)$.

For a fixed rank p, we define the expansion operation exp_p on alphabets, trees, terms, and automata. The symbols in $exp_p(\Sigma)$ will be indexed by priorities, and we will add a new symbol "or" of arity $p + 1$:

$$exp_p(\Sigma) = \{b^r : b \in \Sigma, r = 0, \dots, p\} \cup \{or\} .$$

The priority of or is 0, and that of b^r is r: so $pr(or) = 0$, and $pr(b^r) = r$.

The expansion operation on trees, shown in Figure 3, replaces every node labeled b by a subtree, copying the subtrees of the node:

$$exp_p(b(t_1, \dots, t_{ar(b)})) = \\ or(b^0(t'_1, \dots, t'_{ar(b)}), \dots, b^p(t'_1, \dots, t'_{ar(b)}))$$

where $t'_i = exp_p(t_i)$, for $i = 1, \dots, ar(b)$.

There is the corresponding operation on terms. The term $exp_p(M)$ is obtained from M by replacing every constant b by $\lambda x_1, \dots, x_{ar(b)}.\ or(b^0 x_1 \dots x_{ar(b)}) \dots (b^p x_1 \dots x_{ar(b)})$. We have that for every λY-term $BT(exp_p(M)) = exp_p(BT(M))$.

The expansion operation on automata modifies the transition function, and the priorities. Given $\mathcal{A} = \langle Q, \Sigma, \{\delta_b\}_{b \in \Sigma}, \Omega : Q \to \{0, \dots, p\}\rangle$ we define

$$exp_p(\mathcal{A}) = \\ \langle Q, exp_p(\Sigma), \{\delta'_b\}_{b \in exp_p(\Sigma)}, pr : exp_p(\Sigma) \to \{0, \dots, p\}\rangle$$

where the priority function pr is the one of $exp_p(\Sigma)$. The transition function is:

$\delta'(q, b^r) = \delta(q, b)$	if $\Omega(q) = r$
$\delta'(q, b^r) = \emptyset$	if $\Omega(q) \neq r$
$\delta'(q, or) = \{(\emptyset, \dots, \{q\}, \dots, \emptyset)\}$	$\{q\}$ is on $\Omega(q)$'th position

Proposition 12 Fix a maximal priority p. For every parity automaton $\mathcal{A}$ over an alphabet Σ using only priorities up to p, the visibly parity automaton $exp_p(\mathcal{A})$ over the priority alphabet $exp_p(\Sigma)$ is such that for every closed λY-term M of type o we have:

$BT(M)$ is accepted by $\mathcal{A}$ from q iff
$BT(exp_p(M))$ is accepted by $exp_p(\mathcal{A})$ from q.

The above fact says that modulo exp_p translation, visibly parity automata are equivalent to parity automata.

IV. Recognizability by automata and models

Our main result is that visibly parity automata correspond to finitary powerset models in exactly the same way that automata with trivial acceptance conditions correspond to models with GFP-semantics.

Recall the correspondence between automata and models from Definitions 4 and 5. We can extend it to visibly parity automata. Let us fix an alphabet Σ^{pr} of constants with priorities. From a visibly parity automaton $\mathcal{A}$ we construct a model $\mathcal{D}^{\mathcal{A}}$ as before since the model does not depend on the acceptance condition. From a model $\mathcal{D}$ we construct an automaton $\mathcal{A}_{\mathcal{D}}$ also as before, but now we take the parity condition given by Σ^{pr}. (Recall $\mathrm{up}(\mathcal{A})$, as in Definition 2, accepts the same trees as $\mathcal{A}$.)

Fact 13 Let Σ^{pr} be an alphabet with priorities, and Σ the same alphabet with priorities erased. For every visibly parity automaton $\mathcal{A}$ over Σ^{pr}, and every finitary powerset model $\mathcal{D}$ over Σ:

$$\mathcal{A}_{\mathcal{D}^{\mathcal{A}}} \text{ is } \mathrm{up}(\mathcal{A}), \text{ and } \mathcal{D}^{\mathcal{A}_{\mathcal{D}}} \text{ is } \mathcal{D}.$$

The main result of the paper states that for λY-calculus with priorities the recognizing powers of finitary powerset models, and visibly parity automata are the same. Because of the above fact, an analogous formulation but starting from the model is also true.

Theorem 14 *Let Σ^{pr} be an alphabet with priorities. Let $\mathcal{A}$ be visibly parity automaton over Σ^{pr}, and let $\mathcal{D}^{\mathcal{A}}$ the corresponding powerset model. For every closed parity typable term M of type o:*

$$[\![M]\!]^{\mathcal{D}^{\mathcal{A}}} = \{q : \mathcal{A} \textit{ accepts } BT(M) \textit{ from } q\} \ .$$

Remark: Recall that our parity automata are $\perp$-blind (cf. page 3). This seems like a strange restriction, but in the light of Theorem 14 this is a property of the semantics of terms. One may wonder what makes it that $\perp$ is always accepted, and not always rejected. This can be traced to the axiom $\Gamma, x = (0, \theta) \vdash x : \theta$ of priority types. This axiom makes 0 the neutral priority. If we started priorities from 1, and adopted the same axiom but with 1, then $\perp$ would be always rejected.

To prove the theorem we need to make a link between the semantics of the λ-calculus with priorities and the acceptance of Böhm trees by visibly parity automata. For this we need to understand how a Böhm tree is constructed. We adapt the method from [9] based on Krivine machines. Below we define the a game $K(M, \mathcal{D}^{\mathcal{A}}, q)$ so that we have the following proposition.

Proposition 15 Fix a priority alphabet Σ^{pr}. Consider a visibly parity automaton $\mathcal{A}$ over Σ^{pr}, and the associated model $\mathcal{D}^{\mathcal{A}}$. For every closed priority typable term M of type o over Σ^{pr}, and every state q of $\mathcal{A}$, we have:

$$\mathcal{A} \text{ accepts } BT(M) \text{ from } q \text{ iff Eve wins in } K(M, \mathcal{D}^{\mathcal{A}}, q) \ .$$

With this proposition at hand, to prove Theorem 14 it remains to make a link between winning in $K(M, \mathcal{D}, q)$ and the semantics of M in $\mathcal{D}$. This is the main technical result of this paper.

Theorem 16 *Consider an alphabet with priorities Σ^{pr} and Σ obtained by erasing priorities. Take a finitary powerset model $\mathcal{D}$ over Σ and the base set $\mathcal{P}(Q)$. For every $q \in Q$ and every closed priority typed term M of type o over Σ^{pr}:*

$$q \in [\![M]\!]^{\mathcal{D}} \qquad \textit{iff} \qquad \textit{Eve wins in } K(M, \mathcal{D}, q) \ .$$

In the remaining of this section we will describe the game $K(M, \mathcal{D}, q)$. We outline the proof of Theorem 16 in the next section.

The intuition behind $K(M, \mathcal{D}^{\mathcal{A}}, q)$ is presented in Figure 4. A configuration of a game is of a form $q \leq (N, \rho, S)$ where q is a state of $\mathcal{A}$, and (N, ρ, S) is a configuration of the Krivine machine. In the game, first a head normal-form of a term is computed (if it exists) using the rules of the Krivine machine; this is symbolized by a dashed line in the figure. At that moment a player, called Eve, chooses a transition of the automaton on b, and another player, called Adam, chooses on state and direction in exactly the same way as in the definition of acceptance of a tree by an automaton. This leads to a new configuration, say $q' \leq (K_2, \rho_2, \varepsilon)$ in Figure 4, and the process repeats.

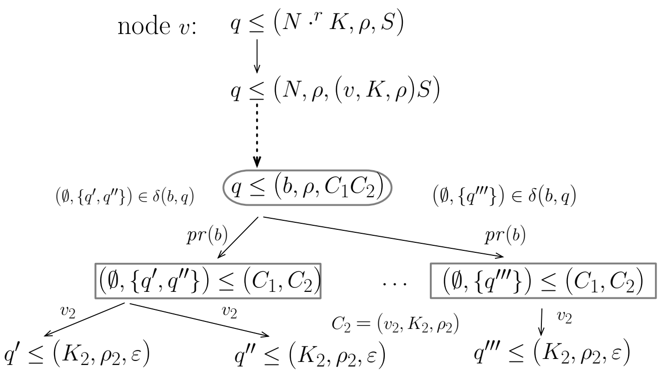

node v:
$q \leq (N \cdot^r K, \rho, S)$
$q \leq (N, \rho, (v, K, \rho)S)$
$q \leq (b, \rho, C_1 C_2)$
$(\emptyset, \{q', q''\}) \in \delta(b, q)$
$(\emptyset, \{q'''\}) \in \delta(b, q)$
$pr(b)$
$pr(b)$
$(\emptyset, \{q', q''\}) \leq (C_1, C_2)$
$\dots$
$(\emptyset, \{q'''\}) \leq (C_1, C_2)$
v_2
v_2
$C_2 = (v_2, K_2, \rho_2)$
v_2
$q' \leq (K_2, \rho_2, \varepsilon)$
$q'' \leq (K_2, \rho_2, \varepsilon)$
$q''' \leq (K_2, \rho_2, \varepsilon)$

Fig. 4. Game $K(M, \mathcal{D}, q_0)$. Eve chooses in rounded boxes, and Adam in rectangular boxes.

We present the game in detail. For the rest of this section we fix a priority typable closed term M of type o, a finitary powerset model $\mathcal{D}$ over the base set $\mathcal{P}(Q)$, and an element $q_0 \in Q$.

First, we will need some terminology and notation related to Krivine machines. A Krivine machine works with environments and closures. The definition of these two concepts is mutually recursive. *Environments*, denoted ρ, are functions from variables to closures. *Closures*, denoted C, are triples (v, N, ρ), where N is a term, ρ is an environment, and v is a node of $K(M, \mathcal{D}, q_0)$ we will construct. Having v in the closure is not standard; we use it to track where the closure was created. As we will see in the rules below, a node v labeled by $q \leq (N \cdot_r K, \rho, S)$ will have a unique successor labeled $q \leq (N, \rho, (v, K, \rho) \cdot S)$ where the closure (v, K, ρ) is created. We write $pr(v)$ to denote r, namely the priority associated to the application in v. A closure can be also created when v is labeled by $q \leq (Y^r F.N, \rho, S)$ and we write $pr(v)$ to denote r in the superscript of Y. We will use $pr(v)$ to state the main invariant of the tree $K(M, \mathcal{D}, q_0)$ with respect to priorities. We say that v is the node of the closure $C = (v, K, \rho)$ and $pr(v)$ is its priority. It will be handy to write $v(C)$ for v, and $pr(C)$ for $pr(v)$.

Definition 17 The *game* $K(M, \mathcal{D}, q_0)$ is played on the tree whose root is labeled by $q_0 \leq (M, \emptyset, \varepsilon)$; where $\emptyset$ is the empty environment, and ε is the empty stack. The tree is constructed according to the rules presented in Figure 5: if l is a label of a node v and $l \xrightarrow{r} l'$ then v has a successor v' labeled l' and r is the label on the edge from v to v'. A label can be a priority or a node; there may be also no label. There are two players, Eve and Adam, who repeatedly choose successors in order to construct an infinite path. Eve chooses a successor in nodes with configurations of the form $(b, \dots)$, Adam chooses a successor in nodes with configurations of the form $(d_1, \dots, d_k) \leq (C_1, \dots, C_k)$. All other nodes have at most one successor. If one of the players cannot make a move she looses. Otherwise the result of a play is an infinite path; Eve wins the play iff the maximal priority seen infinitely often on the path is even.

Let us go back to Figure 4 to see on an example how the game is constructed. In node v, the application rule is used, then the dashed line represents the use of other rules till the head term becomes a constant. At that point the constant rule is used, and it is Eve who chooses a transition, and Adam who chooses a direction and a state. In the example he can only choose the second direction, as there were no states in the first direction. A transition where constant rule is used, is labeled by the priority of the constant. A transition when a closure is used is labeled by a node (the name of the closure).

V. Proof of Theorem 16, an outline

We present an outline of the proof of Theorem 16. The proof consists of three main steps. First, we prove that a certain invariant holds in $K(M, D, d_0)$. This is where priority types are essential. Next, we show a rather straightforward characterization of the semantics of λY-terms with priorities in terms of a game $SG(M, \mathcal{D})$. Finally, we show that the two games are equivalent. This also follows by simple examination of the rules, thanks to the notion of residual form [9].

- $q \leq (\lambda x.N, \rho, C \cdot S) \longrightarrow q \leq (N, \rho[C/x], S)$
- $q \leq (b, \rho, C_1 \dots C_{ar(b)}) \xrightarrow{pr(b)}$ $(d_1, \dots, d_{ar(b)}) \leq (C_1, \dots, C_{ar(b)})$, for $(d_1, \dots, d_{ar(b)})$ such that $q \in [\![b]\!]^{\mathcal{D}}(d_1, \dots, d_{ar(b)})$.
- $(d_1, \dots, d_{ar(b)}) \leq (C_1, \dots, C_{ar(b)}) \xrightarrow{v_i} q_i \leq (K_i, \rho_i, \varepsilon)$ for $q_i \in d_i$, $C_i = (v_i, K_i, \rho_i)$, and $i \in \{1, \dots, ar(b)\}$.
- $q \leq (N \cdot_r K, \rho, S) \longrightarrow q \leq (N, \rho, (v, K, \rho)S)$ v is the node of $q \leq (N \cdot_r K, \rho, S)$.
- $q \leq (Y^r F.N, \rho, S) \longrightarrow q \leq (N, \rho[(v, Y^r F.N, \rho)/F], S)$; v is the node of $q \leq (Y^r F.N, \rho, S)$.
- $q \leq (x, \rho, S) \xrightarrow{v} q \leq (K_v, \rho_v, S)$ where $\rho(x) = (v, K_v, \rho_v)$.

Fig. 5. Rules of constructing $K(M, \mathcal{D}, q_0)$.

A. Priority invariant in $K(M, \mathcal{D}, d_0)$

The whole mechanism of priority types is set up in order to state and guarantee an invariant on the maximal priority between the positions where the closure was created and where the closure was used. To formulate this property we needed to introduce additional parameters v in closures, and on the labels of transitions.

For a node v and its descendant v' in $K(M, \mathcal{D}, d)$, we denote by $pr(v, v')$ the maximal priority appearing on the path from v to v'. Recall that $pr(v)$ stands for the priority of the closure created at v; this is defined by the priority of the application symbol or fixpoint symbol of the term in v.

Lemma 18 (Priority invariant) Game $K(M, \mathcal{D}, d_0)$ satisfies the following priority invariant:

if the unique incoming transition to v' is labeled by v then $pr(v, v') = pr(v)$.

The priority invariant is illustrated in Figure 6. In node v a closure is created because of an application. Then the closure is moved to the environment, because of an abstraction. Later, in v', the closure v is used: a computation makes a look up for a value of a variable x that is bound to the closure created in v. Note that at this moment the environment, the state, and the stack could have changed. The invariant says that the priority of the closure determines the maximal priority seen from the creation to a usage of the closure. Observe that a closure can be used several times.

B. A game characterization of the semantics

Recall that we are working with finitary powerset models as in Definition 3. Instead of taking just any lattice as a base set,

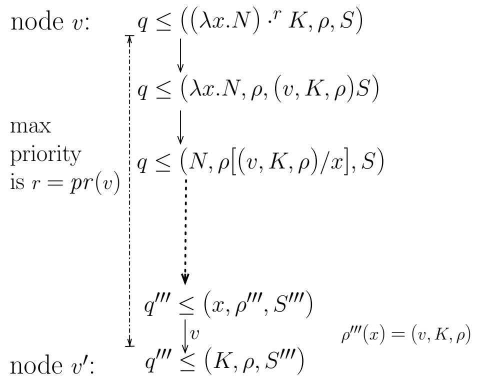

node v:
$q \le ((\lambda x.N) \cdot^r K, \rho, S)$
$q \le (\lambda x.N, \rho, (v, K, \rho)S)$
max priority is $r = pr(v)$
$q \le (N, \rho[(v, K, \rho)/x], S)$
$q''' \le (x, \rho''', S''')$
v
$\rho'''(x) = (v, K, \rho)$
node v':
$q''' \le (K, \rho, S''')$

Fig. 6. Priority invariant in game $K(M, \mathcal{D}, q_0)$.

we have insisted that the base set is the powerset lattice $\mathcal{P}(Q)$ for some set Q. We will use this in the game characterization of the semantics presented in this section. The characterization is a quite direct translation of the semantic clauses into a game. It could have worked for any lattice model, but the distributivity property gives a smoother presentation and will allow later for better complexity arguments.

We will use a notion of a step function that is not completely standard. A *step function of type* $A_1 \to \cdots \to A_k \to o$ is given by $\vec{g} = (g_1, \dots, g_k) \in D_{A_1} \times \cdots \times D_{A_k}$ and $q \in Q$; it is a function $\vec{g} \rightharpoonup q$ such that $(\vec{g} \rightharpoonup d)(h_1, \dots, h_k) = \{q\}$ if $h_i \ge g_i$ for all $i = 1, \dots, k$, and $(\vec{g} \rightharpoonup d)(h_1, \dots, h_k) = \emptyset$ otherwise. A standard notion of a step function would allow any $d \in D_o = \mathcal{P}(Q)$ and not just $q \in Q$. In our notion we allow only atoms of D_o as values. It should be clear that every step function in the standard sense is a supremum of our step functions.

Positions of the game will be of the form $q \le (N, \vartheta, \vec{g})$ where: $q \in Q$ is a state, N is a term, ϑ is a valuation of free variables in N, and $\vec{g}$ is a sequence of elements of the model of appropriate types: if the type of N is $A_1 \to \dots A_k \to o$, then $\vec{g}$ is a sequence of k-elements of type $A_1, \dots, A_k$ respectively. This way $[\![N, \vartheta]\!]$ applied to $\vec{g}$ is an element of D_o. The intuitive meaning of a node $q \le (N, \vartheta, \vec{g})$ is that $q \in [\![N, \vartheta]\!]\vec{g}$.

We define a game $SG(M, \mathcal{D})$ for a closed term M of type o, and model $\mathcal{D}$ over the base set $\mathcal{P}(Q)$. The rules of the game are presented in Figure 7. Eve chooses f in fixpoint and application nodes. Next, Adam chooses a successor in nodes of the form $q \le (f; \dots)$. An infinite play is won by Eve iff the smallest priority seen infinitely often on the edges of the path is even. Actually, a short inspection of the game shows that the size of the term in the first component never increases. This means that $SG(M, \mathcal{D})$ is actually a weak parity game, so it would be enough to use two priorities 0 and 1.

Lemma 19 Consider the game $SG(M, \mathcal{D})$. A position $q \le (N, \vartheta, \vec{g})$ is winning for Eve iff $[\![N, \vartheta]\!] \ge \vec{g} \rightharpoonup q$.

C. Equivalence of $K(M, \mathcal{D}, q_0)$ and $SG(M, \mathcal{D})$.

We prove that the same player wins in the $K(M, \mathcal{D}, q_0)$ as in $SG(M, \mathcal{D})$.

- $q \le (\lambda x.N, \vartheta, f \cdot \vec{g}) \longrightarrow q \le (N, \vartheta[f/x], \vec{g})$
- $$\begin{tikzcd} q \le (Y^r F.N, \vartheta, \vec{g}) \arrow[r] & q \le (f; Y^r F.N, \vartheta, \vec{g}) \arrow[dl] \arrow[d, "r"] \\ q \le (N, \vartheta[f/F], \vec{g}) & q' \le (Y^r F.N, \vartheta, \vec{h}) \end{tikzcd}$$
  for all f and $\vec{h}$ of an appropriate type, and $q' \in f(\vec{h})$.
- $$\begin{tikzcd} q \le (N \cdot_r K, \vartheta, \vec{g}) \arrow[r] & q \le (f; N \cdot_r K, \vartheta, \vec{g}) \arrow[dl] \arrow[d, "r"] \\ q \le (N, \vartheta, f \cdot \vec{g}) & q' \le (K, \vartheta, \vec{h}) \end{tikzcd}$$
  for all f and $\vec{h}$ of an appropriate type, and $q' \in f(\vec{h})$.
- $q \le (b, \vartheta, \vec{g})$ is winning for Eve if $q \in [\![b]\!](\vec{g})$; otherwise Adam wins in this node.
- $q \le (x, \vartheta, \vec{g})$ is winning for Eve iff $q \in \vartheta(x)(\vec{g})$.

Fig. 7. Rules of the game $SG(M, \mathcal{D})$.

Suppose Eve has a winning strategy σ in $K(M, \mathcal{D}, q_0)$. A strategy for Eve in $SG(M, \mathcal{D})$ should tell her what values f to play in application and fixpoint rules. We show how to read them from σ.

We define a *residual* for every closure (v, K_v, ρ_v) created in $K(M, \mathcal{D}, q_0)$. It will be an element of D_A where A is the type of K_v. We denote it by $R^\sigma(v)$. This notation makes use of the fact that v uniquely determines all other elements of the closure. The definition of $R^\sigma(v)$ is on the order of the (simple) type of the closure, namely the order of the type of K_v.

For K_v of type o, we look at all the nodes reachable from v while Eve plays the strategy σ. We select all those who have an incoming transition labeled v. Their labels are necessarily of the form $q' \le (K_v, \rho_v, \varepsilon)$, for some q'. We define $R^\sigma(v)$ to be the set of all such states q'. Observe that since K_v is of type o, the stack in the configuration $(K_v, \rho_v, \varepsilon)$ must be empty.

For K_v of type $A_1 \to \cdots \to A_k \to o$, we also collect collect all the nodes reachable from v when Eve plays σ. We select once again those nodes v' who have incoming edges labeled v. This time the label of v' must have the form $q' \le (K_v, \rho_v, S_{v'})$, for some q' and $S_{v'}$. By typability, $S_{v'}$ is a sequence of closures $C_1, \dots, C_k$ of types $A_1, \dots, A_k$, respectively. By induction $R^\sigma(v(C_1)), \dots, R^\sigma(v(C_k))$ are defined. We consider the step function $(R^\sigma(v(C_1)), \dots, R^\sigma(v(C_k)) \rightharpoonup q'$. We define $R^\sigma(v)$ as the supremum of all such step functions.

Lemma 20 If Eve wins in $K(M, \mathcal{D}, q_0)$ then Eve wins in $SG(M, \mathcal{D})$ from $q_0 \le (M, \emptyset, \varepsilon)$. Moreover she can win by playing with residuals. Analogously for Adam.

This completes the proof of Theorem 16. If Eve wins in

$K(M, \mathcal{D}, q_0)$ then she wins from $q_0 \leq (M, \emptyset, \varepsilon)$ in $SG(M, \mathcal{D})$ and so $q \in \llbracket M, \emptyset \rrbracket^{\mathcal{D}}$ by Lemma 19. Analogously for Adam.

VI. Expressiveness of the λY-calculus with priorities

In this section we show that λY-calculus with priorities is sufficiently expressive: for every assignment of priorities to constants and for every λY-term there is an equivalent λY-term with priorities. By equivalent we mean that the two terms generate the same Böhm trees. The construction of the λY-term with priorities is effective. The presented construction gives an exponentially bigger term, but by sharing common subterms one can obtain a translation with only a quadratic blowup. Anyway the blow-up in the term size is not the main factor in our complexity considerations.

The translation presented below was proposed by Melliès [11], [17]. Here, it is extended to a fixpoint operator. The other translations in the higher-order model checking literature, [15], [28], [29] or even before [30], are bit different. They make a "product" of a term and a finite automaton/model; roughly they work on a normal form without first calculating one. For example, they can be used for so called global model checking problem, or to produce an image under a tree transducer [28]. Melliès' construction handles priorities priorities between a binding and a use of a variable.

Fix an alphabet with priorities, Σ^{pr}. This means that every constant b in Σ^{pr} has its arity $ar(b)$ and its priority $pr(b)$. The two determine a priority type θ_b of b; (cf. page 5). By forgetting priorities we get a normal alphabet Σ, where every constant has a simple type A_b obtained by erasing priorities from θ_b. Let p be the largest priority of a constant in Σ^{pr}. Consider an operation transforming simple types into types with priorities:

$$o^+ = o \qquad (A \to B)^+ = (p, A^+) \to \cdots \to (0, A^+) \to B^+$$

We describe a matching operation on terms. It uses variables with superscripts that correspond to priorities. So for every variable x in the original term, we have $x^0, \ldots, x^p$ in the translated term.

The translation presented in Figure 8 uses some notation. For a term N with variables with superscripts, and a rank i we define $N{\Rsh}_r$ to be a term obtained from N by replacing every free variable x^i in N by $x^{i \oplus r}$; recall that $\oplus$ denotes maximum operation. We will also need a variant of this operation, $N{\Rsh}_{r,F}$, where ${\Rsh}_r$ is applied to all variables but F. For example in $(ax^0F^0){\Rsh}_{r,F}$ is ax^rF^0. Observe that $N{\Rsh}_0$ is just N but sometimes we will still use ${\Rsh}_0$ for consistency.

The translation for a variable just selects variable with priority 0. The translation for a constant is a λ-term that multiplies the arity of the constant by $p+1$, and then selects only components corresponding to the priority of the constant. The translation for the abstraction replicates the abstraction $(p+1)$-times; intuitively x^i corresponds to appearances of x with application priority i (cf. Definition 9). The translation of application duplicates the argument $(p+1)$-times, and uses an application of a different priority for each of the arguments.

$$\begin{aligned}
&\mathsf{pa}(x) = x^0\\
&\mathsf{pa}(b) = \lambda x_1^p \ldots x_1^0 \ldots \lambda x_{ar(b)}^p \ldots x_{ar(b)}^0 .\\
&\qquad\qquad\qquad (\ldots (b \cdot_r x_1^r) \cdots) \cdot_r x_{ar(b)}^r \quad \text{where } r = pr(b)\\
&\mathsf{pa}(\lambda x.N) = \lambda x^p \ldots \lambda x^0 .\mathsf{pa}(N)\\
&\mathsf{pa}(MN) = (\cdots(\mathsf{pa}(M) \cdot_p \mathsf{pa}(N){\Rsh}_p) \cdot_{p-1} \mathsf{pa}(N){\Rsh}_{p-1}) \cdots \cdot_0 \mathsf{pa}(N){\Rsh}_0\\
&\mathsf{pa}(YF.N) = \llparenthesis YF.N \rrparenthesis^0 \quad \text{where}\\
&\llparenthesis YF.N \rrparenthesis^p = Y^pF^p \ldots Y^0F^0.\mathsf{pa}(N){\Rsh}_{p,F}\\
&\llparenthesis YF.N \rrparenthesis^{p-1} = Y^{p-1}F^{p-1} \ldots Y^0F^0.\ \mathsf{pa}(N){\Rsh}_{(p-1),F}\ [\llparenthesis YF.N \rrparenthesis^p / F^p]\\
&\qquad\vdots\\
&\llparenthesis YF.N \rrparenthesis^0 = Y^0F^0.\mathsf{pa}(N){\Rsh}_{0,F}\ [\llparenthesis YF.N \rrparenthesis^1 / F^1 \ldots \llparenthesis YF.N \rrparenthesis^p / F^p]
\end{aligned}$$

Fig. 8. Translation to priority typable terms.

The translation for the fixpoint is by far the most complicated. It uses an auxiliary translation $\llparenthesis YF.N \rrparenthesis^i$.

Remark: It would be tempting to translate $YF.N$ to

$$Y^kF^k.(\ldots(Y^1F^1.(Y^0F^0.\mathsf{pa}(N)){\Rsh}_1){\Rsh}_2)\cdots){\Rsh}_k$$

Unfortunately, the result may be not priority typable. This translation would be typable using the fixpoint rule without the side condition. As we have seen in the example on page 6, without the side condition the rule does not ensure that terms are priority homogeneous which is crucial for our constructions.

The correctness of the translation is stated in the next theorem. In the proof it is very handy to use the equivalence between models and automata with trivial acceptance conditions, Proposition 7.

Theorem 21 *For every closed term M of type o of λY-calculus, term $\mathsf{pa}(M)$ is priority typable, and $BT(\mathsf{pa}(M)) = BT(M)$.*

VII. Higher-order model-checking through powerset models

We examine how we can use the link between automata and models to do higher-order model-checking. Given λY-term M and parity automaton $\mathcal{A}$, we want to decide if $BT(M)$ is accepted by $\mathcal{A}$ from q.

We show that there is no overhead in reducing the higher-order model-checking to evaluation in models. At the same time, the examples we give here show that evaluation in models should not be done naively, by just taking the semantic clauses.

Let us first look at the case when we have a prioritized alphabet Σ^{pr}, a priority typed term M of type o, and a visibly parity automaton $\mathcal{A}$, both over Σ^{pr}. In this case we can construct a model $\mathcal{D}^{\mathcal{A}}$ as in Definition 4. Theorem 14 tells us that $\llbracket M \rrbracket^{\mathcal{D}^{\mathcal{A}}}$ is the set of states q from which $\mathcal{A}$ accepts $BT(M)$. So the model-checking problem reduces to calculating the value of a term in the finitary powerset model constructed from the automaton.

The model-checking problem for λY-calculus, can be reduced to that for λY-calculus with priorities thanks to Proposition 12. Suppose we are given an alphabet Σ of typed constants, a λY-term M, and a parity automaton $\mathcal{A}$; both over the alphabet Σ. For pr the priority function of $\mathcal{A}$, we consider the maximal priority p, and construct a priority alphabet $exp_p(\Sigma)$ (cf. page 6). Both $exp_p(M)$ and $exp_p(\mathcal{A})$ are over the alphabet $exp_p(\Sigma)$, and $exp_p(\mathcal{A})$ is a visibly parity automaton. By Proposition 12, $BT(M)$ is accepted by $\mathcal{A}$ from q, iff $BT(exp_p(M))$ is accepted by $exp_p(\mathcal{A})$ from q. Finally, we can use Theorem 21 to obtain $\mathsf{pa}(exp_p(M))$, a priority typable term with the same Böhm tree as $exp_p(M)$. So the model checking problem reduces to checking if the Böhm tree of $\mathsf{pa}(exp_p(M))$ is accepted by $exp_p(\mathcal{A})$. By Theorem 14, this in turn can be done by evaluating $\mathsf{pa}(exp_p(M)))$ in the model constructed from $exp_p(\mathcal{A})$.

We claim that the complexity of this approach is not worse than that of other approaches to the model checking problem. To carry out the complexity analysis we need to name some parameters of the problem. We have a fixed alphabet of constants with priorities, Σ^{pr}. We use p for the maximal priority in Σ^{pr}. We use $|M|$ for the size of the term, and $|Q|$ for the number of states in $\mathcal{A}$. Let $n > 0$ be the maximal order of the type of a subterm of M; Let $n_{\mathit{fix}} \leq n$ be the maximal order of a fixpoint subterm of M. We start counting the order from 0, namely: $order(o) = 0$, and $order(A \to B) = \max(order(A)+1, order(B))$. Finally, we use K for the maximal arity of a subterm of M; where the arity of a term is the sum of the number of its free variables and the number of its arguments.

Before calculating the complexity, let us remark that the translation from the λY-calculus to the λY-calculus with priorities does not induce an important complexity blowup. The size of $exp_p(\mathcal{A})$ is the same as that of $\mathcal{A}$. The size of $\mathsf{pa}(exp_p(M))$ is $\mathcal{O}(|M| \cdot p^{|M|})$, and its arity is $p \cdot K$. Actually, by encoding common subterms one can get a translation of size quadratic in $p \cdot |M|$, but anyway the size of the term is not a dominant factor in the complexity.

Thus the complexity of the algorithm comes from checking $q \in [\![M]\!]^{\mathcal{D}^{\mathcal{A}}}$. By Lemma 19, to decide $q \in [\![M]\!]^{\mathcal{D}^{\mathcal{A}}}$ we need to find out if Eve has a winning strategy from the position $q \leq (M, \emptyset, \varepsilon)$ in the game $SG(M, \mathcal{D}^{\mathcal{A}})$. The latter is a weak parity game, so in order to establish the complexity of deciding the winner we need to know its size.

We calculate the size of $SG(M, \mathcal{D}^{\mathcal{A}})$. Positions of the game are of the form $q \leq (N, \vartheta, \vec{g})$ or $q \leq (f; N, \vartheta, \vec{g})$; where f is an element of $\mathcal{D}^{\mathcal{A}}$, ϑ is a valuation in $\mathcal{D}^{\mathcal{A}}$, and $\vec{g}$ is a sequence of elements of $\mathcal{D}^{\mathcal{A}}$. By examining the rules of the game $SG(M, \mathcal{D}^{\mathcal{A}})$ we can see that the type of f has order $\leq \max(n-1, n_{\mathit{fix}})$, and hereditary arity $\leq K$. Similarly for elements in $\vec{g}$. The type of the element $\vartheta(x)$ is determined by the type of x. Its order is trivially bounded by n, but when M is closed then it is bounded by $\max(n-1, n_{\mathit{fix}})$, and it has hereditary arity $\leq K$. Thus the orders of f, ϑ, and $\vec{g}$ are bounded by $n_{max} = \max(n-1, n_{\mathit{fix}}) \leq n$. The number of step functions in $\mathcal{D}^{\mathcal{A}}_A$ for a type A of order n and hereditary arity $\leq K$ is bounded by $Tower_n(\mathcal{O}(K|Q|))$. The number of elements in $D^{\mathcal{A}}_A$, is one exponent bigger.

With these calculations we see that there are at most $|M|\, Tower_{n_{max}+1}(\mathcal{O}(K|Q|))$ positions in the game $SG(M, \mathcal{D}^{\mathcal{A}})$. Since the game is a weak parity game, it can be solved in linear time wrt. the number of transitions. So the size of the game gives also the complexity of the algorithm. This is in some respect better than the known algorithms since p does not appear in the $Tower$ term. The reason is that we have considered the problem for priority λY-calculus. For λY-calculus we need to take into account the increase of arity due to $\mathsf{pa}(exp_p(M))$ translation. This gives the complexity $\mathcal{O}(|M| \cdot p^{|M|})\, Tower_{n_{max}+1}(\mathcal{O}(Kp|Q|))$ as do other methods for the λY-calculus [9].

Model-checking higher-order recursive schemes

To look at the complexity of model-checking schemes, we need to look at a translation from schemes to the λY-calculus [31]. Terms obtained by translating schemes are in a β-normal form (but, of course, not in $\beta\delta$-normal form). Moreover, all fixpoint subterms are *semi-closed*: the only free variables are those that are later closed with a fixpoint operator. The notion of arity we have used above becomes a standard one for schemes, since the right-hand sides of equations do not have free variables. If we use the translation from [31] followed by the method described above we do not get the algorithm of same complexity as [8]. The problem is that in op. cit. the algorithm has the complexity of $Tower_n$ while our calculation gives the complexity of $Tower_{n_{max}+1}$. The complexity is bigger when $n_{\mathit{fix}} = n$.

This discrepancy in the complexity is actually not that surprising. The target of our reduction is a weak parity game while the target of the reduction in [8] is a parity game. The problem comes from the fact that in the semantic game, in the case of the fixpoint rule, Eve is required to play with what she thinks approximates the semantics of the fixpoint. One exponent can be saved by limiting her choice: we may allow her to play only with approximations of the fixpoint from the Knaster-Tarski theorem. Their number is bounded by the height of the lattice, so in our case it is one exponent smaller than the size of the lattice. Yet, even better is to handle fixpoints through a parity condition.

We describe a game $PSG(M, \mathcal{D})$ that is a variant of $SG(M, \mathcal{D})$ where fixpoints are handled through unfolding and a parity condition. We assume that every fixpoint subterm of M is semi-closed. Recall that terms obtained from translations of schemes have this property. Without loss of generality we may assume that every fixpoint variable in M is bound once. So a variable F bound in M uniquely identifies the fixpoint subterm $Y^r F.N$ in M. We refer to this subterm as $term(F, M)$.

The rules of the game $PSG(M, \mathcal{D})$ are the same as $SG(M, \mathcal{D})$ (cf. Figure 7) but for those handling the fixpoint. They become:

- $q \leq (Y^r F.N, \vartheta, \vec{g}) \longrightarrow q \leq (N, \vartheta, \vec{g})$

- $q \leq (F, \vartheta, \vec{g}) \xrightarrow{r} q \leq (N, \vartheta, \vec{g})$ when $term(F, M) = Y^r F.N$

The winning condition in $PSG(M, \mathcal{D})$ is the parity condition given on the parities written on the edges. We get an analog of Lemma 20.

Lemma 22 If Eve wins in $K(M, \mathcal{D}, q_0)$ then Eve wins in $PSG(M, \mathcal{D})$ from $q_0 \leq (M, \emptyset, \varepsilon)$. Moreover she can win by playing with residuals. Analogously for Adam.

The size of the game $PSG(M, \mathcal{D}^{\mathcal{A}})$ is of order of magnitude $Tower_n$, since unlike in $SG(M, \mathcal{D}^{\mathcal{A}})$ the sizes of domains for fixpoints do not enter into computation. Thus using $PSG(M, \mathcal{D}^{\mathcal{A}})$ we obtain the same worst case complexity as algorithms working directly for schemes.

Model-checking for disjunctive automata

We show how to do model-checking for disjunctive automata in $(n-1)$-EXPTIME. This result has been proved by Kobayashi and Ong [32]. It is technically very interesting because it is difficult to prove without going into internals of a decision procedure for higher-order model-checking. In our case we will use the game $PSG(M, \mathcal{D})$ and the fact that in this game Eve may play only with residuals. It is this later fact that is difficult to capture on the level of semantics.

A *disjunctive automaton* is a parity automaton whose transition function has the property: for every $(S_1 \dots, S_{ar(b)}) \in \delta_b$, the union $S_1 \cup \dots \cup S_{ar(b)}$ is a singleton. In particular, at most one of $S_1, \dots, S_{ar(b)}$ is not empty. The dual of a disjunctive automaton is a deterministic automaton, potentially exponentially bigger. Observe that if $\mathcal{A}$ is disjunctive then $exp_p(\mathcal{A})$ is also disjunctive. In the light of the above discussion, to get $(n-1)$-EXPTIME algorithm it is enough to show it for λY-calculus with priorities and disjunctive visibly parity automata.

Let us look at $K(M, \mathcal{D}^{\mathcal{A}}, q_0)$ when $\mathcal{A}$ is a disjunctive visibly parity automaton. A winning strategy for Eve in this game is a path. Indeed, branching for Adam appears only at nodes of the form $(d_1, \dots, d_{ar(b)}) \leq (C_1, \dots, C_{ar(b)})$. Because of disjunctiveness, Adam has no choice there. In consequence, every closure of type o is used at most once when Eve is playing her strategy. Indeed, when a v-closure is used in v' due to the transition $\xrightarrow{v} (K_v, \rho_v, \varepsilon)$ then v-closure cannot appear in ρ_v, and the stack must be empty as K_v is a term of type o. So there cannot be any use of the v-closure below v'.

By Lemma 22, Eve can win in $PSG(M, \mathcal{D})$ when playing with residuals coming from a winning strategy σ for Eve in $K(M, \mathcal{D}, q_0)$. By the preceding paragraph, for every closure (v, K_v, ρ_v) with K_v of type o, the residual $R^\sigma(v)$ is a singleton or an empty set. So it is an element of $D_0^{thin} = \{\{q\} :\in Q\} \cup \{\emptyset\}$. By definition of residuals, a residual of a type $A_1 \to \dots \to A_k \to o$ is a set of step functions from $D_{A_1}^{thin} \times \dots \times D_{A_k}^{thin}$ to D_o^{thin}. Hence the size of D_A^{thin}, for a type A of order n, is bounded by $Tower_n(\mathcal{O}(K|Q|))$, compared to $Tower_{n+1}(\mathcal{O}(K|Q|))$ for D_A. So the size of the game $PSG(M, \mathcal{D})$ is of order of magnitude $Tower_{n-1}$, and it can be solved in time exponential in the number of priorities.

VIII. CONCLUSIONS

This work pursues a model-based approach to higher-order model-checking. It proposes an extension of the λY-calculus with priorities and shows that its semantics is perfectly suited for higher-order model-checking, in a sense that there is a correspondence between models and visibly parity automata (Fact 13), such that value in the model coincides with acceptance by the corresponding automaton (Theorem 14). This gives a partial answer to the most fundamental question about the model-based approach, namely is there a simple semantically defined class of models recognizing exactly properties expressed in monadic second-order logic.

The answer is partial since it concerns only λY-calculus with priorities, and is restricted to $\perp$-blind parity automata. Yet, λY-calculus with priorities is sufficiently expressive, as it generates the same Böhm trees as λY-calculus. Moreover, Theorem 14 says that $\perp$-blindness is unavoidable if we want to stay with the interpretation with least and greatest fixpoints.

There exist models that can recognize $\perp$-insightful properties [12], [15], but they are substantially more complicated. The easiest way around seems to simply assume that terms are productive, i.e., their Böhm trees do not have $\perp$. Every term can be transformed to a productive term [15], [33], but the transformation is algorithmically expensive. Instead, one may simply add a new constant in front of every fixpoint operator: the resulting term would be productive, and in its Böhm tree one could see the unfoldings of fixpoints. Observe that already in the propositional mu-calculus guardedness is a technical issue [34].

From a more general perspective, models have a rich structure, and this can guide refinement of the syntax to make this structure explicit. Development of linear logic and differential calculus are flagship examples of this approach. On a much more modest scale, we have followed the same methodology here. We have extracted priorities from models to the syntax, capturing the interactions between computation and priorities in a form of a type system.

Models are modular, and agnostic to syntax. One can extend the syntax as long as it can be interpreted in the model. They may be useful in the context of modular model-checking [35]. It would be interesting to extend the current work to linear constructs investigated recently by Clairambault, Grellois and Murawski [36]. Observe that the size of domains for linear types indicates that it could be possible to recover their complexity results through the model approach.

This work was inspired by the paper of Kobayashi et. al. [19] studying the relation with model checking of higher-order fixpoint logic (HFL-MC). The reduction to λY-calculus with priorities gives a reduction of higher-order model-checking problem to HFL-MC. Except for fixpoints, this is the same reduction as in [19]. It would be very interesting to find an inverse reduction that preserves the structure of fixpoints, depends only on the nesting of fixpoints and not the size of the transition system. A recent paper of Kobayashi, Tsukada, and Watanabe [37] makes a strong case for HFL-MC.

References

[1] N. Kobayashi, “Model checking higher-order programs,” *J. ACM*, vol. 60, no. 3, p. 20, 2013.

[2] T. Tsukada and N. Kobayashi, “Untyped recursion schemes and infinite intersection types,” in *FOSSACS’10*, ser. LNCS, vol. 6014, 2010, pp. 343–357.

[3] A. Murase, T. Terauchi, N. Kobayashi, R. Sato, and H. Unno, “Temporal verification of higher-order functional programs,” in *POPL’16*, 2016, pp. 57–68.

[4] Y. Nanjo, H. Unno, E. Koskinen, and T. Terauchi, “A fixpoint logic and dependent effects for temporal property verification,” in *LICS’18*, 2018, pp. 759–768.

[5] H. Unno, Y. Satake, and T. Terauchi, “Relatively complete refinement type system for verification of higher-order non-deterministic programs,” *PACMPL*, vol. 2, no. POPL, pp. 12:1–12:29, 2018.

[6] C.-H. L. Ong, “On model-checking trees generated by higher-order recursion schemes,” in *LICS*, 2006, pp. 81–90.

[7] M. Hague, A. S. Murawski, C. L. Ong, and O. Serre, “Collapsible pushdown automata and recursion schemes,” *ACM Trans. Comput. Log.*, vol. 18, no. 3, pp. 25:1–25:42, 2017.

[8] N. Kobayashi and L. Ong, “A type system equivalent to modal mu-calculus model checking of recursion schemes,” in *LICS*, 2009, pp. 179–188.

[9] S. Salvati and I. Walukiewicz, “Krivine machines and higher-order schemes,” *Inf. Comput.*, vol. 239, pp. 340–355, 2014.

[10] T. Tsukada and C. L. Ong, “Compositional higher-order model checking via ω-regular games over böhm trees,” in *CSL-LICS ’14*. ACM, 2014, p. 78.

[11] C. Grellois and P. Melliès, “Finitary semantics of linear logic and higher-order model-checking,” in *MFCS’15*, ser. LNCS, vol. 9234, 2015, pp. 256–268.

[12] S. Salvati and I. Walukiewicz, “A Model for Behavioural Properties of Higher-order Programs,” in *CSL’15*, ser. LIPIcs, vol. 41, 2015, pp. 229–243.

[13] S. Salvati, “Recognizability in the simply typed lambda-calculus,” in *WoLLIC*, ser. LNCS, vol. 5514, 2009, pp. 48–60.

[14] S. Salvati and I. Walukiewicz, “Evaluation is MSOL-compatible,” in *FSTTCS 2013*, ser. LIPIcs, vol. 24, 2013, pp. 103–114.

[15] ——, “Using models to model-check recursive schemes,” *Logical Methods in Computer Science*, vol. 11, no. 2, 2015.

[16] K. Aehlig, “A finite semantics of simply-typed lambda terms for infinite runs of automata,” *Logical Methods in Computer Science*, vol. 3, no. 1, pp. 1–23, 2007.

[17] P. Melliès, “Higher-order verification,” June, 2014, workshop on Abstraction and Verification in Semantics. A part of IHP semester on Semantics of proofs and certified mathematics (Paris,France).

[18] N. Kobayashi and C.-H. L. Ong, “Complexity of model checking recursion schemes for fragments of the modal mu-calculus,” *Logical Methods in Computer Science*, vol. 7, no. 4, 2011.

[19] N. Kobayashi, É. Lozes, and F. Bruse, “On the relationship between higher-order recursion schemes and higher-order fixpoint logic.” in *POPL’17*, 2017, pp. 246–259.

[20] C. Grellois, “Semantics of linear logic and higher-order model-checking. (sémantique de la logique linéaire et ”model-checking” d’ordre supérieur),” Ph.D. dissertation, Paris Diderot University, France, 2016.

[21] F. Bruse, “Alternating parity krivine automata,” in *MFCS*, ser. LNCS, vol. 8634, 2014, pp. 111–122.

[22] P. Melliès, “Higher-order parity automata,” in *LICS*, 2017, pp. 1–12.

[23] R. Suzuki, K. Fujima, N. Kobayashi, and T. Tsukada, “Streett automata model checking of higher-order recursion schemes,” in *FSCD’ 2017*, ser. LIPIcs, vol. 84, 2017, pp. 32:1–32:18.

[24] M. Hague, R. Meyer, and S. Muskalla, “Domains for higher-order games,” in *MFCS’17*, ser. LIPIcs, vol. 83, 2017, pp. 59:1–59:15.

[25] M. Hague, R. Meyer, S. Muskalla, and M. Zimmermann, “Parity to safety in polynomial time for pushdown and collapsible pushdown systems,” in *MFCS’18*, ser. LIPIcs, vol. 117, 2018, pp. 57:1–57:15.

[26] I. Walukiewicz, “LambdaY-Calculus With Priorities,” HAL, Tech. Rep., Apr. 2019, https://hal.archives-ouvertes.fr/hal-02100196.

[27] R. M. Amadio and P.-L. Curien, *Domains and Lambda-Calculi*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[28] N. Kobayashi, K. Matsuda, and A. Shinohara, “Functional programs as compressed data,” in *Workshop on Partial Evaluation and Program Manipulation, PEPM 2012*. ACM, 2012, pp. 121–130.

[29] A. Haddad, “Model checking and functional program transformations,” in *FSTTCS*, ser. LIPIcs, vol. 24, 2013, pp. 115–126.

[30] S. van Bakel, “The heart of intersection type assignment: Normalisation proofs revisited,” *Theor. Comput. Sci.*, vol. 398, no. 1-3, pp. 82–94, 2008.

[31] S. Salvati and I. Walukiewicz, “Simply typed fixpoint calculus and collapsible pushdown automata,” *Mathematical Structures in Computer Science*, vol. 26, no. 7, pp. 1304–1350, 2016.

[32] N. Kobayashi and C.-H. L. Ong, “Complexity of model checking recursion schemes for fragments of the modal mu-calculus,” *Logical Methods in Computer Science*, vol. 7, no. 4, 2011.

[33] A. Haddad, “IO vs OI in higher-order recursion schemes,” in *FICS*, ser. EPTCS, vol. 77, 2012, pp. 23–30.

[34] F. Bruse, O. Friedmann, and M. Lange, “On guarded transformation in the modal μ-calculus,” *Logic Journal of the IGPL*, vol. 23, no. 2, pp. 194–216, 2015.

[35] R. Sato and N. Kobayashi, “Modular verification of higher-order functional programs,” in *ESOP’17*, ser. LNCS, vol. 10201, 2017, pp. 831–854.

[36] P. Clairambault, C. Grellois, and A. S. Murawski, “Linearity in higher-order recursion schemes,” *PACMPL*, vol. 2, no. POPL, pp. 39:1–39:29, 2018.

[37] N. Kobayashi, T. Tsukada, and K. Watanabe, “Higher-order program verification via HFL model checking,” in *ESOP’18*, ser. LNCS, vol. 10801, 2018, pp. 711–738.