Using models to model-check recursive schemes

S. Salvati and I. Walukiewicz*

Université de Bordeaux, INRIA, CNRS, LaBRI UMR5800

Abstract. We propose a model-based approach to the model checking problem for recursive schemes. Since simply typed lambda calculus with the fixpoint operator, λY-calculus, is equivalent to schemes, we propose the use a model of λY to discriminate the terms that satisfy a given property. If a model is finite in every type, this gives a decision procedure. We provide a construction of such a model for every property expressed by automata with trivial acceptance conditions and divergence testing. Such properties pose already interesting challenges for model construction. Moreover, we argue that having models capturing some class of properties has several other virtues in addition to providing decidability of the model-checking problem. As an illustration, we show a very simple construction transforming a scheme to a scheme reflecting a property captured by a given model.

1 Introduction

In this paper we are interested in the relation between the effective denotational semantics of the simply typed λY-calculus and the logical properties of Böhm trees. By *effective denotational* semantics we mean semantic spaces in which the denotation of a term can be computed; in this paper, these effective denotational semantics will simply be finite models of the λY-calculus, but Y will often be interpreted neither as the least nor as the greatest fixpoint.

Understanding properties of Böhm trees from a logical point of view is a problem that arises naturally in the model checking of higher-order programs. Often this problem is presented in the context of higher-order recursive schemes that generate a possibly infinite tree. Nevertheless, higher-order recursive schemes can be represented faithfully by λY-terms, in the sense that the infinite trees they generate are precisely Böhm trees of the λY-terms.

The technical question we address is whether the Böhm tree of a given term is accepted by a given tree automaton. We consider only automata with trivial acceptance conditions which we call *TAC automata*. The principal technical challenge we address here is that we allow automata to detect if a term has a head normal form. We call such automata *insightful* as opposed to *Ω-blind* automata that are insensitive to divergence. For example, the models studied by Aehlig or Kobayashi [1,10] are Ω-blind. Considering safety properties and divergence at the same time poses serious challenges to representing with denotational semantics what it means for an automaton to accept a Böhm tree. Indeed, this

* This work has been supported by ANR 2010 BLAN 0202 01 FREC

requires one to give to non-convergence a non-standard interpretation that can influence the meaning of a term in a stronger way than the usual semantics does. As we show here, Y combinator cannot be interpreted as an extremal fixpoint in this case, so known algorithms for verification of safety properties cannot take non-convergence into account in a non-trivial way.

Let us explain the difference between insightful and Ω-blind conditions. The definition of a Böhm tree says that if the head reduction of a term does not terminate then in the resulting tree we get a special symbol Ω. Yet this is not how this issue is treated in all known solutions to the model-checking problem. There, instead of reading Ω the automaton is let to run on the infinite sequence of unproductive reductions. In the case of automata with trivial conditions, this has as an immediate consequence that such an infinite computation is accepted by the automaton. From a denotational semantics perspective, this amounts to interpreting the fixpoint combinator Y as a greatest fixpoint on some finite monotonous model. So, for example, with this approach to semantics, the language of schemes that produce at least one head symbol is not definable by automata with trivial conditions. Let us note that this problem disappears once we consider Büchi conditions as they permit one to detect an infinite unproductive execution. So here we look at a particular class of properties expressible by Büchi conditions. Thus, the problem we address is a non-trivial extension of what is usually understood as the safety property for recursive schemes.

Our starting point is the proof that the usual methods for treating the safety properties of higher-order schemes cannot capture the properties described with insightful automata. The first result of the paper shows that extremal fixpoint models can only capture boolean combinations of Ω-blind TAC automata. Our main result is the construction of a model capturing insightful automata. This construction is based on an interpretation of the fixpoint operator which is neither the greatest nor the least one. The main difficulty is to obtain a definition that guaranties the existence and uniqueness of the fixpoint at every type.

In our opinion providing models capturing certain classes of properties is an important problem both from foundational and practical points of view. On the theoretical side, models need to handle all the constructions of the λ-calculus while, for example, the type systems proposed so far by Kobayashi [10], and by Kobayashi and Ong [13] do not cater for λ-abstraction. In consequence the model-based approach gives more insight into the solution. On the practical side, models capturing classes of properties set the stage to define algorithms to decide these properties in terms of evaluating λ-terms in them. One can remark that models offer most of the algorithmic advantages as other approaches, as illustrated by [16] which shows that the typing discipline of [10] can be completely rephrased in terms of simple models. This practical interest of models has been made into a slogan by Terui [20]: *better semantics, faster computation.* To substantiate further the interest of models we also present a straightforward transformation of a scheme to a scheme reflecting a given property [4]. From a larger perspective, the model based approach opens a new bridge between λ-calculus and model-checking communities. In particular the model we construct for in-

sightful automata brings into the front stage particular non-extremal fixpoints. To our knowledge these were not much studied in the λ-calculus literature.

Related work The model checking problem has been solved by Ong [14] and subsequently revisited in a number of ways [8,13,17]. A much simpler proof for the same problem in the case of Ω-blind TAC automata has been given by Aehlig [1]. In his influential work, Kobayashi [10,9,11] has shown that many interesting properties of higher-order recursive programs can be analyzed with recursive schemes and Ω-blind TAC automata. He has also proposed an intersection type system for the model-checking problem. The method has been applied to the verification of higher-order programs [12,5]. Let us note that at present all algorithmic effort concentrates on Ω-blind TAC automata. In a recent work Ong and Tsukada [15] provide a game semantics model corresponding to Kobayashi's style type system. Their model can handle only Ω-blind automata, but then it is fully complete. We cannot hope to have full completeness in our approach using simple models. In turn, as we mention in [21] and show here, handling Ω-blind automata with simple models is straightforward. The reflection property for schemes has been proved by Broadbent et. al. [4]. Haddad gives a direct transformation of a scheme to an equivalent scheme without divergent computations [7].

Organization of the paper The next section introduces the objects of our study: λY-calculus and automata with trivial acceptance conditions (TAC automata). In the following section we briefly present the correspondence between models of λY with greatest fixpoints and boolean combinations of Ω-blind TAC automata. In Section 4 we give the construction of the model for insightful TAC automata. The last section presents a transformation of a term into a term reflecting a given property. All the missing proofs can be found in the long version of the paper [19].

2 Preliminaries

We introduce two basic objects of our study: λY-calculus and TAC automata. We will look at λY-terms as mechanisms for generating infinite trees that then are accepted or rejected by a TAC automaton. The definitions we adopt are standard ones in the λ-calculus and automata theory. The only exceptions are the notions of a tree signature used to simplify the presentation, and of Ω-blind/insightful automata that are specific to this paper.

2.1 λY-calculus and models

The *set of types* $\mathcal{T}$ is constructed from a unique *basic type* 0 using a binary operation $\to$. Thus 0 is a type and if α, β are types, so is $(\alpha \to \beta)$. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

A *signature*, denoted Σ, is a set of typed constants, that is symbols with associated types from $\mathcal{T}$. We will assume that for every type $\alpha \in \mathcal{T}$ there are constants ω^α, Ω^α and $Y^{(\alpha\to\alpha)\to\alpha}$. A constant $Y^{(\alpha\to\alpha)\to\alpha}$ will stand for a fixpoint

operator. Both ω^α and Ω^α will stand for undefined, but we will need two such constants in Section 4. Of special interest to us will be *tree signatures* where all constants other than Y, ω and Ω have order at most 2. Observe that types of order 2 have the form $0 \to 0 \to \cdots \to 0 \to 0$.

Proviso: To simplify the notation we will suppose that all the constants in a tree signature are either of type 0 or of type $0 \to 0 \to 0$. So they are either a constant of the base type or a function of two arguments over the base type. This assumption does not influence the results of the paper.

The set of *simply typed λ-terms* is defined inductively as follows. A constant of type α is a term of type α. For each type α there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ that are also terms of type α. If M is a term of type β and x^α a variable of type α then $\lambda x^\alpha.M$ is a term of type $\alpha \to \beta$. Finally, if M is of type $\alpha \to \beta$ and N is a term of type α then (MN) is a term of type β. We shall use the usual convention about dropping parentheses in writing λ-terms and we shall write sequences of λ-abstractions $\lambda x_1 \ldots . \lambda x_n . M$ with only one λ: $\lambda x_1 \ldots x_n . M$; moreover when the sequence of abstracted variables is irrelevant we shall write $\lambda \boldsymbol{x}.M$ for a sequence of variables $\boldsymbol{x}$.

The usual operational semantics of the λ-calculus is given by β-contraction. To give the meaning to fixpoint constants we use δ-contraction ($\to_\delta$).

$$(\lambda x.M)N \to_\beta M[N/x] \qquad YM \to_\delta M(YM).$$

We write $\to^*_{\beta\delta}$ for the $\beta\delta$-reduction, the reflexive and transitive closure of the sum of the two relations. Given a term $M = \lambda x_1 \ldots x_n . N_0 N_1 \ldots N_p$ where N_0 is of the form $(\lambda x.P)Q$ or YP, then N_0 is called the *head redex* of M. We write $M \to_{\beta\delta h} M'$ when M' is obtained by $\beta\delta$-contracting the head redex of M (when it has one). We write $\to^*_{\beta\delta h}$ and $\to^+_{\beta\delta h}$ respectively for the reflexive and transitive closure and the transitive closure of $\to_{\beta\delta h}$. The relation $\to^*_{\beta\delta h}$ is called *head reduction*. A term with no head redex is said to be in *head normal form*.

It is well known that every term has at most one normal form, but due to δ-reduction there are terms without a normal form. A term is *unsolvable* if it does not have a head normal form; otherwise the term is *solvable*. Observe that even if all the subterms of a term are solvable the reduction may generate an infinitely growing term. It is thus classical in the λ-calculus to consider a kind of infinite normal form that by itself is an infinite tree, and in consequence it is not a term of λY [3,2].

A *Böhm tree* is an unranked, ordered, and potentially infinite tree with nodes labelled by terms of the form $\lambda x_1 \ldots . x_n . N$; where N is a variable or a constant, and the sequence of λ-abstractions is optional. So for example x^0, Ω^0, $\lambda x^0.\omega^0$ are labels, but $\lambda y^0.\ x^{0\to 0} y^0$ is not.

Definition 1. *A* Böhm tree *of a term M is obtained in the following way.*

- *If $M \to^*_{\beta\delta} \lambda \boldsymbol{x}.N_0 N_1 \ldots N_k$ with N_0 a variable or a constant then $BT(M)$ is a tree having the root labelled $\lambda \boldsymbol{x}.N_0$ and having $BT(N_1)$, ..., $BT(N_k)$ as its subtrees.*

– *Otherwise* $BT(M) = \Omega^\alpha$, *where* α *is the type of* M.

Observe that a term M without the constants Ω and ω has a $\beta\delta$-normal form if and only if $BT(M)$ is a finite tree without the constants Ω and ω. In this case the Böhm tree is just another representation of the normal form.

Recall that in a tree signature all constants except of Y, Ω, and ω are of type 0 or $0 \to 0 \to 0$. A closed term without λ-abstraction and Y over such a signature is just a finite binary tree: constants of type 0 occur at leaves and those of type $0 \to 0 \to 0$ occur at internal nodes. The same holds for Böhm trees:

Lemma 1. *If* M *is a closed term of type* 0 *over a tree signature then* $BT(M)$ *is a potentially infinite binary tree.*

We will consider finitary models of λY-calculus. In the first part of the paper we will concentrate on those where Y is interpreted as the greatest fixpoint.

Definition 2. *A* GFP-model *of a signature* Σ *is a tuple* $\mathcal{S} = \langle \{\mathcal{S}_\alpha\}_{\alpha\in\mathcal{T}}, \rho\rangle$ *where* $\mathcal{S}_0$ *is a finite lattice, and for every type* $\alpha \to \beta \in \mathcal{T}$, $\mathcal{S}_{\alpha\to\beta}$ *is the lattice* $\mathrm{mon}[\mathcal{S}_\alpha \to \mathcal{S}_\beta]$ *of monotone functions from* $\mathcal{S}_\alpha$ *to* $\mathcal{S}_\beta$ *ordered coordinatewise. The valuation function* ρ *is required to satisfy certain conditions:*

– *If* $c \in \Sigma$ *is a constant of type* α *then* $\rho(c)$ *is an element of* $\mathcal{S}_\alpha$.
– *For every* $\alpha \in \mathcal{T}$, *both* $\rho(\omega^\alpha)$ *and* $\rho(\Omega^\alpha)$ *are the greatest elements of* $\mathcal{S}_\alpha$.
– *Moreover,* $\rho(Y^{(\alpha\to\alpha)\to\alpha})$ *is the function assigning to every function* $f \in \mathcal{S}_{\alpha\to\alpha}$ *its greatest fixpoint.*

Observe that every $\mathcal{S}_\alpha$ is finite, hence all the greatest fixpoints exist without any additional assumptions on the lattice.

A *variable assignment* is a function υ associating to a variable of type α an element of $\mathcal{S}_\alpha$. If s is an element of $\mathcal{S}_\alpha$ and x^α is a variable of type α then $\upsilon[s/x^\alpha]$ denotes the valuation that assigns s to x^α and that is identical to υ otherwise.

The *interpretation of a term* M of type α in the model $\mathcal{S}$ under the valuation υ is an element of $\mathcal{S}_\alpha$ denoted $[\![M]\!]^\upsilon_{\mathcal{S}}$. The meaning is defined in the standard way: for constants it is given by ρ; for variables by υ; the application is interpreted as function application, and finally for abstraction $[\![\lambda x^\alpha.M]\!]^\upsilon_{\mathcal{S}}$ is a function mapping an element $s \in \mathcal{S}_\alpha$ to $[\![M]\!]^{\upsilon[s/x^\alpha]}_{\mathcal{S}}$. As usual, we will omit subscripts or superscripts in the notation of the semantic function if they are clear from the context.

It is known that Böhm trees are a kind of initial semantics for λ-terms. In particular if two terms have the same Böhm trees then they have the same semantics in every GFP model. To look at it more closely we need to formally define the semantics of a Böhm tree.

The semantics of a Böhm tree is defined in terms of its truncations. For every $n \in \mathbb{N}$, we denote by $BT(M)\downarrow_n$ the finite term that is the result of replacing in the tree $BT(M)$ every subtree at depth n by the constant ω^α of the appropriate type. Observe that if M is closed and of type 0 then α will always be the base type 0. This is because we work with a tree signature. We define:

$$[\![BT(M)]\!]^\upsilon_{\mathcal{S}} = \bigwedge \{ [\![BT(M)\downarrow_n]\!]^\upsilon_{\mathcal{S}} \mid n \in \mathbb{N}\}.$$

The above definitions are standard for λY-calculus, or more generally for PCF [2]. In particular the following proposition, in a more general form, can be found as Exercise 6.1.8 in op. cit.[1]

Proposition 1. *If $\mathcal{S}$ is a finite GFP-model and M is a closed term then: $[\![M]\!]_{\mathcal{S}} = [\![BT(M)]\!]_{\mathcal{S}}$.*

2.2 TAC Automata

Let us fix a tree signature Σ. This means that apart from ω, Ω and Y all constants have order at most 2. Let Σ_0 be the set of constants of type 0, and Σ_2 the set of constants of type $0 \to 0 \to 0$. By Lemma 1, in this case Böhm trees are potentially infinite binary trees.

Definition 3. *A* finite tree automaton with trivial acceptance condition *(TAC automaton) over the signature $\Sigma = \Sigma_0 \cup \Sigma_2$ is*

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta_0 : Q \times (\Sigma_0 \cup \{\Omega\}) \to \{ff, tt\}, \delta_2 : Q \times \Sigma_2 \to \mathcal{P}(Q^2)\rangle$$

where Q is a finite set of states and $q^0 \in Q$ is the initial state. The transition function of TAC automaton may be the subject to the additional restriction:

$$\textbf{\textit{Ω-blind:}} \quad \delta_0(q, \Omega) = tt \text{ for all } q \in Q.$$

Automata satisfying this restriction are called Ω-blind. *For clarity, we use the term* insightful *to refer to automata without this restriction.*

Automata will run on Σ-labelled binary trees that are partial functions $t : \{1,2\}^* \to \Sigma \cup \{\Omega\}$ such that their domain is a binary tree, and $t(u) \in \Sigma_0 \cup \{\Omega\}$ if u is a leaf, and $t(u) \in \Sigma_2$ otherwise.

A *run of $\mathcal{A}$ on t* is a labelling $r : \{1,2\}^* \to Q$ of t such that the root is labeled by q^0 and the labelling of the successors of a node respects the transition function δ. A run is *accepting* if $\delta_0(r(u), t(u)) = tt$ for every leaf u of t . A tree is *accepted by* $\mathcal{A}$ if there is an accepting run on the tree. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of trees accepted by $\mathcal{A}$.

Observe that TAC automata have acceptance conditions on leaves, expressed with δ_0, but do not have acceptance conditions on infinite paths.

As underlined in the introduction all the work on automata with trivial conditions relies on the Ω-blind restriction. Let us give some examples of properties that can be expressed with insightful automata but not with Ω-blind automata.

- The set of terms not having Ω in their Böhm tree. To recognize this set we take the automaton with a unique state q. This state has transitions on all the letters from Σ_2. It also can end a run in every constant of type 0 except for Ω: this means $\delta_0(q, \Omega) = ff$ and $\delta_0(q, c) = tt$ for all other c.

[1] In this paper we work with finite monotone models which are a particular case of the directed complete partial orders used in [2].

- The set of terms having a head normal form. We take an automaton with two states q and $q_\top$. From $q_\top$ automaton accepts every tree. From q it has transitions to $q_\top$ on all the letters from Σ_2, on letters from Σ_0 it behaves as the automaton above.
- Building on these two examples one can easily construct an automaton for a property like "every occurrence of Ω is preceded by a constant *err*".

It is immediate to see that none of these languages can be recognized by a Ω-blind automaton since if such an automaton accepts a tree t then it accepts also every tree obtained by replacing a subtree of t by Ω.

3 GFP models and Ω-blind TAC automata

In this short section we summarize the relation between GFP models and Ω-blind TAC automata. We start with the expected formal definition of the set of λY-terms recognized by a model.

Definition 4. *For a GFP model $\mathcal{S}$ over the base set $\mathcal{S}_0$. The* language recognized *by a subset $F \subseteq \mathcal{S}_0$ is the set of closed λY-terms $\{M \mid [\![M]\!]_{\mathcal{S}} \in F\}$.*

Proposition 2. *For every Ω-blind TAC automaton $\mathcal{A}$, the language of $\mathcal{A}$ is recognized by a GFP model.*

Let $\mathcal{A}$ be an automaton as in Definition 3. For the model $\mathcal{S}$ in question we take a GFP model with $\mathcal{S}_0 = \mathcal{P}(Q)$. This defines $\mathcal{S}_\alpha$ for every type α. It remains to define the interpretation of constants other than ω, Ω, or Y. The meaning of a constant c of type 0 is $\{q \mid \delta_0(q, c) = tt\}$; and the meaning of a of type $0 \to 0 \to 0$ is a function whose value on $(S_0, S_1) \in \mathcal{P}(Q)^2$ is $\{q \mid \delta_2(q, a) \cap S_0 \times S_1 \neq \emptyset\}$. Finally, for the set $F_{\mathcal{A}}$ used to recognize $L(\mathcal{A})$ we will take $\{S \mid q^0 \in S\}$; recall that q^0 is the initial state of $\mathcal{A}$. With these definitions it is possible to show that for every closed term M of type 0: $BT(M) \in L(\mathcal{A})$ iff $[\![M]\!] \in F_{\mathcal{A}}$.

Next theorem shows that the recognizing power of GFP models is actually characterized by Ω-blind TAC automata. The right-to-left implication of this theorem has been stated in [21].

Theorem 1. *A language L of λ-terms is recognized by a GFP-model iff it is a boolean combination of languages of Ω-blind TAC automata.*

Using the results in [16], it can be shown that typings in Kobayashi's type systems [10] give precisely values in GFP models.

4 A model for insightful TAC automata

The goal of this section is to present a model capable of recognizing languages of insightful TAC automata. Theorem 1 implies that the fixpoint operator in such a model can be neither the greatest nor the least fixpoint. In the first subsection

we will construct a model containing at the same time a model with the least fixpoint and a model with the greatest fixpoint. We cannot just take the model generated by the product of the base sets of the two models as we will need that the value of a term in the least fixpoint component influences the value in the greatest fixpoint component. In the second part of this section we will show how to interpret insightful TAC automata in such a model.

4.1 Model construction and basic properties

We are going to construct a model $\mathcal{K}$ intended to recognize the language of a given insightful TAC automaton. This model is built on top of the standard model $\mathcal{D}$ for detecting if a term has a head-normal form.

Consider a family of sets $\{\mathcal{D}_\alpha\}_{\alpha\in\mathcal{T}}$; where $\mathcal{D}_0 = \{\bot, \top\}$ is the two element lattice, and $\mathcal{D}_{\alpha\to\beta}$ is $\mathrm{mon}[\mathcal{D}_\alpha \to \mathcal{D}_\beta]$. So for every α, $\mathcal{D}_\alpha$ is a finite lattice. We shall refer to the minimal and maximal element of $\mathcal{D}_\alpha$ respectively with the notations $\bot_\alpha$ and $\top_\alpha$.

Consider the model $\mathcal{D} = \langle\{\mathcal{D}_\alpha\}_{\alpha\in\mathcal{T}}, \rho\rangle$ where ω and Ω are interpreted as the least elements, and Y is interpreted as the least fixpoint operator. So $\mathcal{D}$ is a dual of a GFP model as presented in Definition 2. The reason for not taking a GFP model here is that we would prefer to use the greatest fixpoint later in the construction. To all constants other than Y, ω, and Ω the interpretation ρ assigns the greatest element of the appropriate type. The following theorem is well-known (cf [2] page 130).

Theorem 2. *For every closed term M of type 0 without ω we have:*

$$BT(M) = \Omega \quad \textit{iff} \quad [\![M]\!]_{\mathcal{D}} = \bot.$$

We fix a finite set Q and $Q_\Omega \subseteq Q$. Later these will be the set of states of a TAC automaton, and the set of states from which the automaton accepts Ω, respectively. To capture the power of such an automaton, we are going to define a model $\mathcal{K}(Q, Q_\Omega, \rho)$ of the λY-calculus with a non-standard interpretation of the fixpoint. Roughly, this model will live inside the product of $\mathcal{D}$ and the GFP model $\mathcal{S}$ for an Ω-blind automaton. The idea is that every set $\mathcal{K}_\alpha$ will have a projection on $\mathcal{D}$ but not necessarily on $\mathcal{S}$. This allows to observe whether a term converges or not, and at the same time to use this information in computing in the second component.

Definition 5. *For a given finite set Q and $Q_\Omega \subseteq Q$ we define a family of sets $\mathcal{K}_{Q,Q_\Omega} = (\mathcal{K}_\alpha)_{\alpha\in\mathcal{T}}$ by mutual recursion together with a family of relations $\mathcal{L} = (\mathcal{L}_\alpha)_{\alpha\in\mathcal{T}}$ such that $\mathcal{L}_\alpha \subseteq \mathcal{K}_\alpha \times \mathcal{D}_\alpha$:*

1. *We let $\mathcal{K}_0 = \{(\top, P) \mid P \subseteq Q\} \cup \{(\bot, Q_\Omega)\}$ with the order: $(d_1, P_1) \leq (d_2, P_2)$ iff $d_1 \leq d_2$ in $\mathcal{D}_0$ and $P_1 \subseteq P_2$. (cf. Figure 1)*
2. *$\mathcal{L}_0 = \{((d, P), d) \mid (d, P) \in \mathcal{K}_0\}$,*
3. *$\mathcal{K}_{\alpha\to\beta} = \{f \in \mathrm{mon}[\mathcal{K}_\alpha \to \mathcal{K}_\beta] \mid \exists d \in \mathcal{D}_{\alpha\to\beta}. \forall (g, e) \in \mathcal{L}_\alpha. (f(g), d(e)) \in \mathcal{L}_\beta\}$,*
4. *$\mathcal{L}_{\alpha\to\beta} = \{(f, d) \in \mathcal{K}_{\alpha\to\beta} \times \mathcal{D}_{\alpha\to\beta} \mid \forall (g, e) \in \mathcal{L}_\alpha. (f(g), d(e)) \in \mathcal{L}_\beta\}$.*

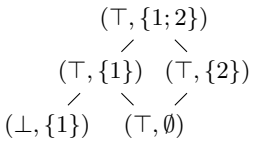

Fig. 1. The order $\mathcal{K}_0$ for $Q = \{1, 2\}$ and $Q_\Omega = \{1\}$

Note that every $\mathcal{K}_\alpha$ is finite since it lives inside the standard model constructed from $\mathcal{D}_0 \times \mathcal{P}(Q)$ as the base set. Moreover for every α, $\mathcal{K}_\alpha$ is a join semilattice and thus has a greatest element. Recall that a TAC automaton is supposed to accept unsolvable terms from states Q_Ω. So the unsolvable terms of type 0 should have Q_Ω as a part of their meaning. This is why $\bot$ of $\mathcal{D}_0$ is associated to $(\bot, Q_\Omega)$ in $\mathcal{K}_0$ via the relation $\mathcal{L}_0$. This also explains why we needed to take the least fixpoint in $\mathcal{D}$. If we had taken the greatest fixpoint then the unsolvable terms would have evaluated to $\top$ and the solvable ones to $\bot$. In consequence we would have needed to relate $\top$ with $(\top, Q_\Omega)$, and we would have been forced to relate $\bot$ with $(\bot, Q)$. But since $(\top, Q_\Omega)$ and $(\bot, Q)$ are incomparable in $\mathcal{K}_0$ we would not have been able to obtain the order preserving injection $(\cdot)^\uparrow$ from $\mathcal{D}_0$ to $\mathcal{K}_0$ that is defined below at every type:

Definition 6. *For every $h \in \mathcal{D}_\alpha$ we define the element $h^\uparrow$ of $\mathcal{K}_\alpha$:*

$$h^\uparrow = \bigvee \{f \mid (f, h) \in \mathcal{L}_\alpha\} \ .$$

It can be shown that this element always exists, and that $(\cdot)^\uparrow$ is a monotone embedding of $\mathcal{D}$ into $\mathcal{K}$. Moreover $(d^\uparrow, d)$ is in $\mathcal{L}_\alpha$ for very $d \in \mathcal{D}_\alpha$. One can also verify that the relation $\mathcal{L}_\alpha$ is functional, so we get the projection operation.

Definition 7. *For every type α and $f \in \mathcal{K}_\alpha$ we let $\overline{f}$ to be the unique element of $\mathcal{D}_\alpha$ such that $(f, \overline{f}) \in \mathcal{L}_\alpha$.*

We are now going to give the definition of the interpretation of the fixpoint combinator in $\mathcal{K}$. This definition is based on the fixpoint operator in $\mathcal{D}$. As a shorthand, we write fix_α for the operation in $\mathcal{D}_{(\alpha\to\alpha)\to\alpha}$ mapping a function of $\mathcal{D}_{\alpha\to\alpha}$ to its least fixpoint. It can be shown that for every $f \in \mathcal{K}_{\alpha\to\alpha}$ the sequence $f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow)$ is decreasing in $\mathcal{K}_\alpha$.

Definition 8. *For every type α and $f \in \mathcal{K}_\alpha$ define*

$$\mathrm{Fix}_\alpha(f) = \bigwedge_{n\in\mathbb{N}} (f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow))$$

We are ready to define the model we were looking for.

Definition 9. *For a finite set Q and $Q_\Omega \subseteq Q$ consider a tuple $\mathcal{K}(Q, Q_\Omega, \rho) = (\mathcal{K}_{Q,Q_\Omega}, \rho)$ where $\mathcal{K}_{Q,Q_\Omega}$ is as in Definition 5 and ρ is a valuation such that for every type α: ω^α is interpreted as the greatest element of $\mathcal{K}_\alpha$, $Y^{(\alpha\to\alpha)\to\alpha}$ is interpreted as Fix_α, and Ω^0 is interpreted as $(\bot, Q_\Omega)$.*

Theorem 3. *The model $\mathcal{K}(Q, Q_\Omega, \rho)$ is a model of the λY-calculus.*

Let us mention the following useful fact showing a correspondence between the meanings of a term in $\mathcal{K}$ and in $\mathcal{D}$. The proof is immediate since, by definition, $\{\mathcal{L}_\alpha\}_{\alpha\in\mathcal{T}}$ is a logical relation (cf [2]).

Lemma 2. *For every type α and closed term M of type α:*

$$(\llbracket M\rrbracket_{\mathcal{K}}, \llbracket M\rrbracket_{\mathcal{D}}) \in \mathcal{L}_\alpha\,.$$

4.2 Correctness and completeness of the model

It remains to show that the model we have constructed can recognize languages of TAC automata. We fix a tree signature Σ and a TAC automaton $\mathcal{A}$ as in Definition 3. So Q is the set of states of $\mathcal{A}$ and Q_Ω is the set of states q such that $\delta(q,\Omega) = tt$. Consider a model $\mathcal{K}$ based on $\mathcal{K}(Q, Q_\Omega, \rho)$ as in Definition 9. We need to specify the meaning of constants like $c : 0$ or $a : 0^2 \to 0$ in Σ:

$$\begin{aligned}\rho(c) =& (\top, \{q : \delta(q,c) = tt\})\\ \rho(a)(d_1,R_1)(d_2,R_2) =& (\top, R) \qquad \text{where } d_1, d_2 \in \{\bot,\top\} \text{ and}\\ & R = \{q \in Q \mid \delta(q,a) \cap R_1 \times R_2 \neq \emptyset\}\end{aligned}$$

It is easy to verify that the meanings of constants are indeed in the model.

Proposition 3. *Given a closed term M of type 0: $BT(M) = \Omega^0$ iff $\llbracket M\rrbracket_{\mathcal{K}} = (\bot, Q_\Omega)$.*

As in the case of GFP-models the semantics of a Böhm tree is defined in terms of its truncations: $\llbracket BT(M)\rrbracket_{\mathcal{K}} = \bigwedge\{\llbracket BT(M)\downarrow_n\rrbracket_{\mathcal{K}} : n \in \mathbb{N}\}$. The subtle, but crucial, difference is that now Ω^0 and ω^0 do not have the same meaning. Nevertheless the analog of Proposition 1 still holds in $\mathcal{K}$.

Theorem 4. *For very closed term M of type 0: $\llbracket M\rrbracket_{\mathcal{K}} = \llbracket BT(M)\rrbracket_{\mathcal{K}}$.*

Proof (Sketch). First we show that $\llbracket M\rrbracket_{\mathcal{K}} \leq \llbracket BT(M)\rrbracket_{\mathcal{K}}$. For this we define a finite approximation of the Böhm tree. *The Abstract Böhm tree up to depth l* of a term M, denoted $ABT_l(M)$, will be a term obtained by reducing M till it resembles $BT(M)$ up to depth l as much as possible. We define $ABT_0(M) = M$, and also $ABT_{l+1}(M) = M$ if M is unsolvable, or otherwise $ABT_{l+1}(M) = \lambda x.N_0 ABT_l(N_1)\ldots ABT_l(N_k)$, where $\lambda x.N_0N_1\ldots N_k$ is the head normal form of M.

Since $ABT_l(M)$ is obtained from M by a sequence of $\beta\delta$-reductions, $\llbracket M\rrbracket_{\mathcal{K}} = \llbracket ABT_l(M)\rrbracket_{\mathcal{K}}$ for every l. It remains to show that for every term M and every l:

$$\llbracket M\rrbracket_{\mathcal{K}} = \llbracket ABT_l(M)\rrbracket_{\mathcal{K}} \leq \llbracket BT(M)\downarrow_l\rrbracket_{\mathcal{K}}.$$

Up to depth l, the two terms have the same tree structure. We check that the meaning of every leaf in $ABT_l(M)$ is not bigger than the meaning of the

corresponding leaf of $BT(M)\downarrow_l$. For leaves of depth l this is trivial since on the one hand we have a term and on the other the constant ω. For other leaves, the terms are either identical or on one side we have an unsolvable term, and on the other Ω^0. By Proposition 3 the two have the same meaning in $\mathcal{S}$.

For the inequality in the other direction observe that if a term M does not have Y combinators, then it is strongly normalizing and the theorem is trivial. So we need be able to deal with Y combinators in M. We introduce new constants c_N for every subterm YN of M. The type of c_N is $\boldsymbol{\alpha} \to \beta$ if β is the type of YN and $\boldsymbol{\alpha} = \alpha_1 \dots \alpha_k$ is the sequence of types of the sequence of free variables $\boldsymbol{x} = x_1 \dots x_k$ occurring in YN. We let the semantics of a constant c_N be

$$[\![c_N]\!]_{\mathcal{K}} = \lambda \boldsymbol{p}. \left(\mathrm{fix}_\beta(\overline{[\![N]\!]_{\mathcal{D}}^{[\boldsymbol{p}/\boldsymbol{x}]}})\right)^{\uparrow} .$$

In the full version of the paper we show that $[\![c_N]\!]$ is in $\mathcal{K}$. Moreover for every $p_1, \dots, p_k, q_1, \dots, q_l$:

$$[\![c_N]\!]_{\mathcal{K}}(p_1, \dots, p_k)(q_1, \dots, q_l) = \begin{cases} (\bot, Q_\Omega) & \text{if } [\![c_N]\!]_{\mathcal{D}}(\overline{p}_1, \dots, \overline{p}_k)(\overline{q}_1, \dots, \overline{q}_l) = \bot \\ (\top, Q) & \text{if } [\![c_N]\!]_{\mathcal{D}}(\overline{p}_1, \dots, \overline{p}_k)(\overline{q}_1, \dots, \overline{q}_l) = \top \end{cases} \tag{1}$$

We now define term $\mathit{iterate}^n(N)$ for very $n \in \mathbb{N}$.

$$\mathit{iterate}^0(N) = c_N \qquad \mathit{iterate}^{n+1}(N) = \lambda \boldsymbol{x}.N(\mathit{iterate}^n(N)\boldsymbol{x}) .$$

From the definition of the fixpoint operator in $\mathcal{K}$ and the fact that $\mathcal{K}_\beta$ is finite it follows that $[\![\mathit{iterate}^n(N)]\!] = [\![\lambda \boldsymbol{x}.YN]\!]$ for some n. Now we can apply this identity to all fixpoint subterms in M starting from the innermost subterms. So the term $\mathit{expand}^i(M)$ is obtained by repeatedly replacing occurrences of subterms of the form YN in M by $\mathit{iterate}^i(N)\boldsymbol{x}$ starting from the innermost occurrences. We get that for n chosen as above $[\![M]\!]_{\mathcal{K}} = [\![\mathit{expand}^n(M)]\!]_{\mathcal{K}}$.

We come back to the proof. The missing inequality will be obtained from

$$[\![M]\!]_{\mathcal{K}} = [\![\mathit{expand}^n(M)]\!]_{\mathcal{K}} = [\![BT(\mathit{expand}^n(M))]\!]_{\mathcal{K}} \geq [\![BT(M)]\!]_{\mathcal{K}} .$$

The first equality we have discussed above. The second is trivial since $\mathit{expand}^n(M)$ does not have fixpoints. It remains to show $[\![BT(\mathit{expand}^n(M))]\!]_{\mathcal{K}} \geq [\![BT(M)]\!]_{\mathcal{K}}$.

Let us denote $BT(\mathit{expand}^n(M))$ by P. So P is a term of type 0 in a normal form without occurrences of Y. For a term K let $\tilde{K}$ be a term obtained from K by simultaneously replacing c_N by $\lambda \boldsymbol{x}.YN$. By definition of the fixpoint we have $[\![c_N]\!]_{\mathcal{K}} \geq [\![\lambda \boldsymbol{x}.YN]\!]_{\mathcal{K}}$ which also implies that $[\![K]\!]_{\mathcal{K}} \geq [\![\tilde{K}]\!]_{\mathcal{K}}$. Moreover, as $\tilde{P} =_{\beta\delta} M$, we have that $BT(\tilde{P}) = BT(M)$. We need to show that $[\![P]\!]_{\mathcal{K}} \geq [\![BT(\tilde{P})]\!]_{\mathcal{K}}$.

Let us compare the trees $BT(P)$ and $BT(\tilde{P})$ by looking on every path starting from the root. The first difference appears when a node v of $BT(P)$ is labelled with c_N for some N. Say that the subterm of P rooted in v is $c_N K_1 \dots K_i$. Then at the same position in $BT(P')$ we have the Böhm tree of the term $(\lambda \boldsymbol{x}.YN)\tilde{K}_1 \dots \tilde{K}_i$. We will be done if we show that $[\![c_N K_1 \dots K_i]\!]_{\mathcal{K}} \geq [\![BT((\lambda \boldsymbol{x}.YN)\tilde{K}_1 \dots \tilde{K}_i)]\!]_{\mathcal{K}}$.

We reason by cases. If $[\![c_N K_1 \dots K_i]\!]_{\mathcal{D}} = \top$ then equation (1) gives us $[\![c_N K_1 \dots K_i]\!]_{\mathcal{K}} = (\top, Q)$. So the desired inequality holds since $(\top, Q)$ is the greatest element of $\mathcal{K}_0$.

If $[\![c_N K_1 \dots K_i]\!]_{\mathcal{D}} = \bot$ then $[\![c_N \tilde{K}_1 \dots \tilde{K}_i]\!]_{\mathcal{D}} = \bot$ since $[\![K_i]\!]_{\mathcal{K}} \geq [\![\tilde{K}_i]\!]_{\mathcal{K}}$. By equation (1) we get $[\![c_N \tilde{K}_1 \dots \tilde{K}_i]\!]_{\mathcal{D}} = (\bot, Q_\Omega)$. Since, by the definition of the fixpoint operator, $[\![c_N]\!]_{\mathcal{K}} \geq [\![\lambda \boldsymbol{x}.\, YN]\!]_{\mathcal{K}}$ we get $[\![YN\tilde{K}_1 \dots \tilde{K}_i]\!]_{\mathcal{K}} = (\bot, Q_\Omega)$. But then Proposition 3 implies that $YNK_1 \dots K_i$ is unsolvable. Thus we get $[\![BT((\lambda \boldsymbol{x}.YN)\tilde{K}_1 \dots \tilde{K}_i)]\!]_{\mathcal{K}} = [\![\Omega]\!]_{\mathcal{K}} = (\bot, Q_\Omega)$. □

Once we know that the semantics of the Böhm tree of a term in the model is the same as the semantics of a term, the proof of the correctness of the model is quite straightforward and very similar to the case of GFP models.

Theorem 5. *Let $\mathcal{A}$ be an insightful TAC automaton and $\mathcal{K}$ a model as at the beginning of the subsection. For every closed term M of type 0:*

$$BT(M) \in L(\mathcal{A}) \quad \textit{iff} \quad q^0 \textit{ is in the second component of } [\![M]\!]_{\mathcal{K}}.$$

5 Reflection

The idea behind the notion of a reflecting term is that at every moment of its evaluation every subterm should know its meaning. Knowing the meaning amounts to extra labelling of constants. Formally, we express this by the notion of a reflective Böhm tree defined below. The definition can be made more general but we will be interested only in the case of terms of type 0. In this section we will show that reflective Böhm trees can be generated by λY-terms.

As usual we suppose that we are working with a tree signature Σ. We will also need a signature where constants are annotated with elements of the model. If $\mathcal{S} = \langle \{\mathcal{S}_\alpha\}_{\alpha \in \mathcal{T}}, \rho \rangle$ is a finitary model then the extended signature $\Sigma^{\mathcal{S}}$ contains constants a^s where a is a constant in $\mathcal{S}$ and $s \in \mathcal{S}_0$; so superscripts are possible interpretations of terms of type 0 in $\mathcal{S}$.

Definition 10. *Let $\mathcal{S}$ be a finitary model and M a closed term of type 0. A* reflective Böhm tree with respect to $\mathcal{S}$ *is obtained in the following way:*

- *If $M \to^*_{\beta\delta} bN_1N_2$ for some constant $b : 0 \to 0 \to 0$ then $rBT_{\mathcal{S}}(M)$ is a tree having the root labelled by $b^{[\![bN_1N_2]\!]_{\mathcal{S}}}$ and having $rBT_{\mathcal{S}}(N_1)$ and $rBT_{\mathcal{S}}(N_2)$ as subtrees.*
- *If $M \to^*_{\beta\delta} c$ for some constant $c : 0$ then $rBT_{\mathcal{S}}(M) = c^{[\![c]\!]_{\mathcal{S}}}$.*
- *Otherwise, M is unsolvable and $BT(M) = \Omega^0$.*

Observe that when $\mathcal{S}$ satisfies $[\![N]\!]_{\mathcal{S}} = [\![BT(N)]\!]_{\mathcal{S}}$ for every term N then the superscripts in $rBT(M)$ are the meanings of respective subtrees in the Böhm tree. When, moreover, $\mathcal{S}$ recognizes a given property then these superscripts determine if the tree satisfies the property. These two conditions are fulfilled by the models we have considered in this paper.

We will use terms to construct reflective Böhm trees.

Definition 11. *Let Σ be a tree signature, and $\mathcal{S}$ a finitary model. Let M be a closed term of type 0 over the signature Σ. We say that a term M' over the signature $\Sigma^{\mathcal{S}}$ is* a reflection of M in $\mathcal{S}$ *if $BT(M') = rBT(M)$.*

The objective of this section is to construct reflections of terms. Since λY-terms can be translated to schemes and vice versa, the construction would work for schemes too. (Translations between schemes and λY-terms that do not increase the type order are presented in [18]).

Let us fix a tree signature Σ and a finitary model $\mathcal{S}$. For the construction of reflective terms we enrich λY-calculus with some syntactic sugar. Consider a type α. The set $\mathcal{S}_\alpha$ is finite for every type α; say $\mathcal{S}_\alpha = \{d_1, \ldots, d_k\}$. We will introduce a new atomic type $[\alpha]$ and constants $d_1 \ldots, d_k$ of this type; there will be no harm in using the same names for constants and elements of the model. We do this for every type α and consider terms over this extended type discipline. Notice that in the result there are no other closed normal terms than $d_1, \ldots, d_k$ of type $[\alpha]$.

Given a term M of type $[\alpha]$ and $M_1, \ldots M_n$ that are all terms of type β, we introduce the construct

$$\text{case } M\{d_i \to M_i\}_{d_i \in \mathcal{S}_\alpha}$$

which is a term of type β and which reduces to M_i when $M = d_i$. This construct is a simple syntactic sugar. We could as well represent $[\alpha]$ as the type $\beta^k \to \beta$, and a constant d_i by the i^{th} projection $\lambda x_1 \ldots x_n.x_i$. We would then get that the term $MM_1 \ldots M_k$ reduces to M_i and thus behaves exactly as the *case* construct.

We define a transformation on types $\alpha^\bullet$ by induction on their structure:

$$(\alpha \to \beta)^\bullet = \alpha^\bullet \to [\alpha] \to \beta^\bullet \quad \text{and} \quad \alpha^\bullet = \alpha \text{ when } \alpha \text{ is atomic.}$$

The translation we are looking for will be an instance of a more general translation $[M, \upsilon]$ of a term M of type α into a term of type $\alpha^\bullet$ where υ is a valuation over $\mathcal{S}$.

$$\begin{aligned}
[\lambda x^\alpha.M, \upsilon] &= \lambda x^{\alpha^\bullet} \lambda y^{[\alpha]}.\ \text{case } y^{[\alpha]}\{d \to [M, \upsilon[d/x^\alpha]]\}_{d \in \mathcal{S}_\alpha} \\
[MN, \upsilon] &= [M, \upsilon]\, [N, \upsilon]\, [\![N]\!]^\upsilon \\
[a, \upsilon] &= \lambda x_1^0 \lambda y_1^{[0]} \lambda x_2^0 \lambda y_2^{[0]}. \\
&\qquad \text{case } y_1^{[0]}\{d_1 \to \text{case } y_2^{[0]}\{d_2 \to a^{\rho(a) d_1\, d_2} x_1 x_2\}_{d_2 \in \mathcal{S}_0}\}_{d_1 \in \mathcal{S}_0} \\
[x^\alpha, \upsilon] &= x^{\alpha^\bullet} \\
\left[Y^{(\alpha\to\alpha)\to\alpha} M, \upsilon\right] &= Y^{(\alpha^\bullet \to \alpha^\bullet)\to\alpha^\bullet}(\lambda x^{\alpha^\bullet}.\, [M, \upsilon]\, x^{\alpha^\bullet} [\![YM]\!]^\upsilon)
\end{aligned}$$

To prove correctness of this translation, we show that a head reduction of the original term can be simulated by a sequence of head reductions.

Lemma 3. *If $M \to_{\beta\delta h} M'$, then $[M, \upsilon] \to^+_{\beta\delta h} [M', \upsilon]$.*

Theorem 6. *For every finitary model $\mathcal{S}$ and a closed term M of type 0:*

$$BT([M, \emptyset]) = rBT_{\mathcal{S}}(M)\ .$$

Remark: If in a model $\mathcal{S}$ the divergence can be observed (as it is the case for GFP models and for the model $\mathcal{K}$, cf. Proposition 3) then in the translation above we could add the rule $[M, \upsilon] = \Omega$ whenever $[\![M]\!]^{\upsilon}_{\mathcal{S}}$ denotes a diverging term. We would obtain a term which would always converge. A different construction for achieving the same goal is proposed in [7].

Remark: Even though the presented translation preserves the structure of a term, it makes the term much bigger due to *case* construction in the clause for λ-abstraction. The blow-up is unavoidable due to complexity lower-bounds on the model-checking problem. Nevertheless, one can try to limit the use of *case* construct. We present a slightly more efficient translation that takes the value of the known arguments into account. For this, the translation also depends on a stack of values from $\mathcal{S}$ in order to recall the values taken by the arguments. For the sake of simplicity, we also assume that the constants always have all their arguments (this can be achieved by using terms in η-long form).

$$
\begin{aligned}
[\lambda x^{\alpha}.M, \upsilon, d :: S] &= \lambda x^{\alpha^{\bullet}} y^{[\alpha]}.\, [M, \upsilon[d/x^{\alpha}], S] \\
[\lambda x^{\alpha}.M, \upsilon, \varepsilon] &= \lambda x^{\alpha^{\bullet}} y^{[\alpha]}.\mathrm{case}\ y^{[\alpha]}\{d \to [M, \upsilon[d/x^{\alpha}], \varepsilon]\}_{d \in \mathcal{S}_{\alpha}} \\
[MN, \upsilon, S] &= [M, \upsilon, [\![N]\!]^{\upsilon} :: S]\, [N, \upsilon, \varepsilon]\, [\![N]\!]^{\upsilon} \\
[a, \upsilon, d_1 :: d_2 :: \varepsilon] &= \lambda x_1^0 \lambda y_1^{[0]} \lambda x_2^0 \lambda y_2^{[0]}.\ a^{[\![a]\!] d_1 d_2} x_1 x_2 \\
[x^{\alpha}, \upsilon, S] &= x^{\alpha^{\bullet}} \\
[YM, \upsilon, S] &= Y\, [M, \upsilon, [\![YM]\!]^{\upsilon} :: S]
\end{aligned}
$$

6 Conclusions

We have extended the scope of the model-based approach to a larger class of properties. While a priori it is more difficult to construct a finitary model than to come up with a decision procedure, in our opinion this additional effort is justified. It allows, as we show here, to use the techniques of the theory of the λ-calculus. It opens new ways of looking at the algorithmics of the model-checking problem. Since typing in intersection type systems [10] and step functions in models are in direct correspondence [16], model-based approach can also benefit from all the developments in algorithms based on typing. Finally, this approach allows to get new constructions as demonstrated by our transformation of a scheme to a scheme reflecting a given property. Observe that this transformation is general and does not depend on our particular model.

Let us note that the model-based approach is particularly straightforward for Ω-blind TAC automata. It uses standard observations on models of the λY-calculus and Proposition 2 with a simple inductive proof. The model we propose for insightful automata may seem involved; nevertheless, the construction is based on simple and standard techniques. Moreover, this model implements an

interesting interaction between components. It succeeds in mixing a GFP model for Ω-blind automaton with the model $\mathcal{D}$ for detecting solvability.

The approach using models opens several new perspectives. One can try to characterize what kinds of fixpoints correspond to what class of automata conditions. More generally, models hint a possibility to have an Eilenberg like variety theory for lambda-terms [6]. This theory would cover infinite regular words and trees too as they can be represented by λY-terms. Finally, considering model-checking algorithms, the model-based approach puts a focus on computing fixpoints in finite partial orders. This means that a number of techniques, ranging from under/over-approximations, to program optimization can be applied.

References

1. K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.
2. R. M. Amadio and P-L. Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.
3. H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
4. C. Broadbent, A. Carayol, L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129, 2010.
5. Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *ICALP (2)*, volume 7392 of *LNCS*, pages 165–176, 2012.
6. S. Eilenberg. *Automata, Languages and Machines*. Academic Press, New York, 1974.
7. A. Haddad. IO vs OI in higher-order recursion schemes. In *FICS*, volume 77 of *EPTCS*, pages 23–30, 2012.
8. M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.
9. N. Kobayashi. Higher-order program verification and language-based security. In *ASIAN*, volume 5913 of *LNCS*, pages 17–23. Springer, 2009.
10. N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.
11. N. Kobayashi. Types and recursion schemes for higher-order program verification. In *APLAS*, volume 5904 of *LNCS*, pages 2–3, 2009.
12. N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FOSSACS*, pages 260–274, 2011.
13. N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.
14. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.
15. C.-H. L. Ong and T. Tsukada. Two-level game semantics, intersection types, and recursion schemes. In *ICALP (2)*, volume 7392 of *LNCS*, pages 325–336, 2012.
16. S. Salvati, G. Manzonetto, M. Gehrke, and H. Barendregt. Loader and Urzyczyn are logically related. In *ICALP (2)*, LNCS, pages 364–376, 2012.
17. S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, volume 6756 of *LNCS*, pages 162–173, 2011.

18. S. Salvati and I. Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In *RP*, volume 7550 of *LNCS*, pages 6–20, 2012.
19. Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. Technical report, LaBRI, 2012. http://hal.inria.fr/hal-00741077.
20. Kazuchige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
21. I. Walukiewicz. Simple models for recursive schemes. In *MFCS*, volume 7464 of *LNCS*, pages 49–60, 2012.