Using models to model-check recursive schemes

Sylvain Salvati, Igor Walukiewicz

▶ **To cite this version:**

Sylvain Salvati, Igor Walukiewicz. Using models to model-check recursive schemes. TLCA 2013, 2013, Eindhoven, Netherlands. pp.189-204. hal-00741077v2

HAL Id: hal-00741077
https://inria.hal.science/hal-00741077v2

Submitted on 12 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using models to model-check recursive schemes

S. Salvati and I. Walukiewicz*

October 12, 2012

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 to 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. We argue that having a model capable of recognizing terms satisfying a given property has other benefits than just providing decidability of the model-checking problem. We show a very simple construction transforming a scheme to a scheme reflecting a given property.

1 Introduction

Recursive schemes are abstract forms of programs where the meaning of constants is not specified. In consequence, the meaning of a scheme is a potentially infinite tree labeled with constants obtained from the unfolding of the recursive definition. Recursive schemes can be also represented in the λY-calculus: simply typed λ-calculus with fixpoint combinators. In this context the meaning of a scheme is just the Böhm tree of a term.

We consider the model checking problem: given a λY-term decide if the Böhm tree it generates is accepted by a given tree automaton with trivial acceptance conditions. The main novelty of the present work is to solve this problem using models for λY-calculus. For a given automaton we construct a finitary model such that the value of a term in this model tells us if the automaton accepts the Böhm tree of the term. The principal technical challenge we address here is that we allow automata to detect if a term has

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

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 [Aeh07] or Kobayashi [Kob09c] are Ω-blind.

As the name suggests automata with trivial acceptance conditions (TAC automata for short) are a very special case of tree automata. In terms of expressive power they can recognize precisely the safety properties. It has been found by Aehlig [Aeh07] that for recursive schemes the model checking problem is conceptually much easier if restricted to automata with trivial conditions. Subsequently Kobayashi has presented a type system for this variant of the model checking problem. This system has been a basis of a successful implementation and application effort [Kob09b, LNOR10, Kob11]. Recall that the decidability of the general case of the model-checking problem has been shown by Ong [Ong06], and then reproved several times using a number of different methods [HMOS08, KO09, SW11]. Yet none of them is as simple as the case of automata with trivial conditions, and none of them has been implemented.

In this paper we look closer at the issue that has been somehow pushed aside in previous works. The semantics of a recursive scheme says that if the scheme gets into an infinite computation without producing a new head symbol 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 case of automata with trivial conditions, this has an immediate consequence that such an infinite computation is accepted by the automaton. So, for example, with this approach to semantics the language of schemes that produces 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 to detect an infinite unproductive execution. So here we look at the particular class of properties expressible by a Büchi condition.

In our opinion solving the model-checking problem using models for λY-calculus is important. First, models need to handle all the constructions of the λ-calculus while, for example, the type systems proposed so far by Kobayashi [Kob09c], and by Kobayashi and Ong [KO09] do not cater for λ-abstraction. In consequence the model-based approach gives more insight into the solution while, from algorithmic point of view, offering the same advantages as typing [SMGB12]. To substantiate we present a straightforward transformation of a scheme to scheme reflecting a given property [BCOS10]. More importantly, the model based approach opens a new bridge between λ-calculus and model-checking communities. In particular the model we

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

The main result of the paper is a construction of a finitary model from a given insightful automaton with trivial acceptance conditions. The meaning of a term in this model determines if the Böhm tree of the term is accepted by the automaton. In this model the fixpoint operator is neither the greatest nor the least one. Indeed, and this is the second result of the paper, models with extremal fixpoints correspond exactly to boolean combinations Ω-blind automata with trivial acceptance conditions. The third result is a transformation of a term into a term reflecting a given property in a sense of [BCOS10]. Roughly, a scheme reflects some property when during its execution it knows if the tree generated from the current point satisfies the property or not. This is a very useful technical property [Bro12], that can also be used in program transformation. The known solutions to this problem are based on a translation to collapsible pushdown automata, so the obtained scheme has no resemblance to the initial one. With the help of models we get a simple and direct translation.

Related work The model checking problem for schemes with respect to monadic second-order logic has been solved by Ong [Ong06] and subsequently revisited in a number of ways [HMOS08, KO09, SW11]. Much simpler proof for the same problem in the case of automata with trivial conditions have been given by Aehlig [Aeh07]. In his influential work, Kobayashi [Kob09c, Kob09a, Kob09d] has shown that many interesting properties of higher-order recursive programs can be analyzed with recursive schemes and automata with trivial conditions. He has also proposed an intersection type system for the model-checking problem. The method has been applied to verification of higher-order programs. In a recent work Ong and Tsukada [OT12] provide a game semantics model corresponding to Kobayashi style proof 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 [Wal12] and show here, handling Ω-blind automata with simple models is straightforward. Reflection property for schemes have been proved by Broadbent et. al. [BCOS10]. Haddad gives a direct transformation of a scheme to an equivalent scheme without divergent computations [Had12].

Organization of the paper We find that the richer syntax of λY-calculus offers more freedom than that of schemes. Yet the two formalism are closely related and there are direct translations between them (cf. [SW12]). The next section introduces objects of our study: λY-calculus and automata with trivial acceptance conditions (TAC automata). In the following section we 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.

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: $\mathit{order}(0) = 1$, and $\mathit{order}(\alpha \to \beta) = \mathit{max}(1 + \mathit{order}(\alpha), \mathit{order}(\beta))$.

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 $0^i \to 0$ for some i; the latter is a short notation for $0 \to 0 \to \cdots \to 0 \to 0$, where there are $i+1$ occurrences of 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.

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 (we write $\to^+_{\beta\delta}$ for its transitive closure). This relation defines an operational equality on terms. We write $=_{\beta\delta}$ for the smallest equivalence relation containing $\to^*_{\beta\delta}$. It is called $\beta\delta$*-conversion* or $\beta\delta$*-equality*. Given a term $M = \lambda x_1 \ldots x_n.N_0N_1 \ldots N_p$ 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 obtain 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 in *head normal form*.

Thus, the operational semantics of the λY-calculus is the $\beta\delta$-reduction. It is well-known that this semantics is confluent and enjoys subject reduction (*i.e.* the type of terms is invariant under computation). So every term has at most one normal form, but due to δ-reduction there are terms without a normal form. A term may not have a normal form because it does not have head normal form, in such case it is called *unsolvable*. Even if a term has a head normal form, i.e. it is *solvable*, it may contain an unsolvable subterm that prevents it from having a normal form. Finally, it may be also the case that all the subterms of a term are solvable but the reduction generates 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 [Bar84, AC98].

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 $M \to^*_{\beta\delta} \lambda\vec{x}.N_0N_1\dots N_k$ with N_0 a variable or a constant then $BT(M)$ is a tree having root labelled by $\lambda\vec{x}.N_0$ and having $BT(N_1)$, $\dots$, $BT(N_k)$ as 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. Unlike in the standard theory of the simply typed λ-calculus we will be rather interested in terms with infinite Böhm trees.

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, where constants of type 0 are in leaves, and constants of type $0 \to 0 \to 0$ are in the internal nodes. The same holds for Böhm trees:

Lemma 2 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 (those with least fixpoints are just dual to these ones).

Definition 3 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 inductively:

- $[\![c]\!]^\upsilon_\mathcal{S} = \rho(c)$
- $[\![x^\alpha]\!]^\upsilon_\mathcal{S} = \upsilon(x^\alpha)$
- $[\![MN]\!]^\upsilon_\mathcal{S} = [\![M]\!]^\upsilon_\mathcal{S}([\![N]\!]^\upsilon_\mathcal{S})$
- $[\![\lambda x^\alpha.M]\!]^\upsilon_\mathcal{S}$ is a function mapping an element $s \in \mathcal{S}_\alpha$ to $[\![M]\!]^{\upsilon[s/x^\alpha]}_\mathcal{S}$ that by abuse of notation we may write $\lambda s.[\![M]\!]^{\upsilon[s/x^\alpha]}_\mathcal{S}$.

As usual, we will omit subscripts or superscripts in the notation of the semantic function if they are clear from the context.

Of course a GFP model is sound with respect to $\beta\delta$-conversion. Hence two $\beta\delta$-convertible terms have the same semantics in the model. For us it is important that a stronger property holds: if two terms have the same Böhm trees then they have the same semantics in the model. For this 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 [AC98]. In particular the following proposition, in a more general form, can be found as Exercise 6.1.8 in op. cit.

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. Recall that this means that apart from ω, Ω and Y all constants have order at most 2. According to our proviso from page 4 all constants in Σ have either type 0 or type $0 \to 0 \to 0$. In this case, by Lemma 2, Böhm trees are potentially infinite binary trees. Let Σ_0 be the set of constants of type 0, and Σ_2 the set of constants of type $0 \to 0 \to 0$.

Definition 4 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:

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

An automaton satisfying this restriction is 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 mapping $r : \{1,2\}^* \to Q$ with the same domain as t an such that:

- $r(\varepsilon) = q^0$, here ε is the root of t.
- $(r(u1), r(u2)) \in \delta_2(t(u), r(u))$ if u is an internal node.

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 previous works on automata with trivial conditions rely on 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.
- 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 easy to see that none of these languages is recognized by 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. This observation also allows to show that those languages cannot be defined as boolean combinations of Ω-blind automata.

3 GFP models and Ω-blind TAC automata

In this section we show that the recognizing power of GFP models coincides with that of boolean combinations of Ω-blind TAC automata. For every automaton we will construct a model capable of discriminating the terms accepted by the automaton. For the opposite direction, we will construct a collection of TAC automata from a model. We start with the expected formal definition of a set of λY-terms recognized by a model.

Definition 5 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 $\{M \mid [\![M]\!]_{\mathcal{S}} \in F\}$.

We now give some notations that we shall use in the course of the proofs. Given a closed term M of type 0, the tree $BT(M)$ can be seen as a binary tree $t : \{1,2\}^* \to \Sigma$. For every node in the domain of t, we write M_v for the subtree of t rooted at node v. The tree $BT(M)\downarrow_k$ is a prefix of this tree containing nodes up to depth k, denote it t_k. Every node v in the domain of t_k corresponds to a subterm of $BT(M)\downarrow_k$ that we denote M_v^k.

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

Proof
For the model $\mathcal{S}_{\mathcal{A}}$ 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. A constant c of type 0 is interpreted as a set $\{q \mid \delta_1(q,c) = tt\}$. A constant a of type $0 \to 0 \to 0$ is interpreted as 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 $\mathcal{A}$. We want to show that for every closed term M of type 0:

$$BT(M) \in L(\mathcal{A}) \quad \text{iff} \quad [\![M]\!] \in F_{\mathcal{A}}.$$

For the direction from left to right, we take a λY-term M such that $BT(M) \in L(\mathcal{A})$, and show that $q^0 \in [\![BT(M)]\!]$. This will do as $[\![BT(M)]\!] = [\![M]\!]$ by Proposition 1. Recall that $[\![BT(M)]\!] = \bigwedge\{[\![BT(M)\downarrow_k]\!] \mid k = 1, 2, \dots\}$. So it is enough to show that $q^0 \in [\![BT(M)\downarrow_k]\!]$ for every k.

Let us assume that we have an accepting run r of $\mathcal{A}$ on $BT(M)$. By induction on the height of v in the domain of $BT(M)\downarrow_k$ we show that $r(v) \in [\![M^k_v]\!]$. The desired conclusion will follow by taking $v = \varepsilon$; that is the root of the tree. If v is a "cut leaf" then M^k_v is ω^0. So $r(v) \in Q = [\![\omega^0]\!]$. If v is a "normal" leaf then M^k_v is a constant c of type 0. We have $r(v) \in \{q : \delta(q, c) = tt\}$. If v is an internal node then $M^k_v = aM^k_{v1}M^k_{v2}$. By induction assumption $r(v1) \in [\![M^k_{v1}]\!]$ and $r(v2) \in [\![M^k_{v2}]\!]$. Hence by definition of $\rho(a)$ we get

$$r(v) \in [\![M_v]\!] = \rho(a)([\![M^k_{v1}]\!], [\![M^k_{v2}]\!]) \ .$$

For the direction from right to left we take a term M and a state $q \in [\![M]\!]$. We construct a run of $\mathcal{A}$ on $BT(M)$ that starts with the state q. So we put $r(\varepsilon) = q$. By Proposition 1, we know that $q \in [\![BT(M)\downarrow_k]\!]$ for all k. There is a letter a such that every term $BT(M)\downarrow_k$ is of the form ω or $aM^k_1M^k_2$. Since $\delta(q, a)$ is a finite set of pairs, there is a pair $(q_1, q_2) \in \delta(q, a)$ such that $q_1 \in [\![M^k_1]\!]$ and $q_2 \in [\![M^k_2]\!]$ for infinitely many k. We repeat the argument with the state q_1 from nodes 1 and with the state q_2 from node 2. It is easy to see that this gives an accepting run of $\mathcal{A}$. □

We are now going to see that the power of GFP models is characterized by Ω-blind TAC automata. We will show that every language recognized by a GFP model is a boolean combination of languages of Ω-blind TAC automata. For the rest of the subsection we fix a tree signature Σ and a GFP model $\mathcal{S} = \langle\{\mathcal{S}_\alpha\}_{\alpha\in\mathcal{T}}, \rho\rangle$ over Σ.

We construct a family of automata that reflect the model $\mathcal{S}$. We let Q be equal to the set $\mathcal{S}_0$. We define $\delta_0 : Q \times \Sigma_0 \cup \{\Omega\} \to \{ff, tt\}$ and $\delta_1 : Q \times \Sigma_2 \to \mathcal{P}(Q^2)$ to be the functions such that:

$$\begin{aligned}\delta_0(q, a) =& tt \quad \text{iff} \quad q \leq \rho(a) \qquad \text{(in the order of } \mathcal{S}_0)\\ \delta_1(q, a) =& \{(q_1, q_2) \mid q \leq \rho(a)(q_1, q_2)\}.\end{aligned}$$

For q in Q, we define $\mathcal{A}_q$ to be the automaton with the starting state q and the other components as above:

$$\mathcal{A}_q = \langle Q, \Sigma, q, \delta_0, \delta_1\rangle$$

We have the following lemma:

Lemma 6 Given a closed λ-term M of type 0: $BT(M) \in L(\mathcal{A}_q)$ iff $q \leq [\![M]\!]$.

Proof
We start by showing that if $\mathcal{A}_q$ accepts $BT(M)$ then $q \leq [\![M]\!]$. Proposition 1 reduces this implication to proving that $q \leq [\![BT(M)]\!]$. Since $[\![BT(M)]\!] = \bigwedge\{[\![BT(M)\downarrow_k]\!] \mid k \in \mathbb{N}\}$, we need to show that for every $k > 0$, $q \leq [\![BT(M)\downarrow_k]\!]$. Fix an accepting run r of $\mathcal{A}_q$ on $BT(M)$. We are going to show that for every v in the domain of $BT(M)\downarrow_k$, $r(v) \leq [\![M_v^k]\!]$. This will imply that $r(\varepsilon) = q \leq [\![BT(M)]\!]\downarrow_k$.

We proceed by induction on the height of v. In case v is a "cut leaf" then M_v^k is ω^0 and $[\![M_v^k]\!]$ is the greatest element of $\mathcal{S}_0$ so that $r(v)$ is indeed smaller than $[\![M_v^k]\!]$. In case v is a "normal leaf" then M_v^k is a constant c of type 0. Since r is an accepting run, we need to have, by definition, $r(v) \leq \rho(c) = [\![M_v^k]\!]$. In case v is an internal node then $M_v^k = aM_{v1}^k M_{v2}^k$, and, by induction, we have that $r(vi) \leq [\![M_{vi}^k]\!]$. Moreover, because r is a run, we need to have $r(v) \leq \rho(a)(r(v1))(r(v2))$, but since $\rho(a)$ is monotone, and $r(vi) \leq [\![M_{vi}^k]\!]$, we have $\rho(a)(r(v1))(r(v2)) \leq \rho(a)([\![M_{v1}^k]\!])([\![M_{v2}^k]\!]) = [\![M_v^k]\!]$. This proves, as expected, that $r(v) \leq [\![M_v^k]\!]$.

Now given $q \leq [\![M]\!]$ we are going to construct a run of $\mathcal{A}_q$ on $BT(M)$. For a node v of $BT(M)$ let M_v denote the subtree rooted in this node. Take r defined by $r(v) = [\![M_v]\!]$ for every v. We show that r is a run of the automaton $\mathcal{A}_{[\![M]\!]}$. Since $q \leq [\![M]\!]$, by the definitions of δ_0 andδ_1, this run can be easily turned into a run of $\mathcal{A}_q$.

By definition $r(\varepsilon) = [\![M]\!] = [\![BT(M)]\!]$. In case v is a leaf c, then $r(v) = \rho(c)$ and we have $\delta_0(c, \rho(c)) = tt$. In case v is an internal node labeled by a, then, by definition $[\![M_v]\!] = \rho(a)([\![M_{v1}]\!], [\![M_{v2}]\!])$, so $([\![M_{v1}]\!], [\![M_{v2}]\!])$ is in $\delta_1(a, [\![M_v]\!])$. □

This lemma and Proposition 2 allow us to infer the announced correspondence.

Theorem 7 *A language L of λ-terms is recognized by a GFP-model iff it is a boolean combination of languages of TAC automata.*

Proof
For the left to right direction take a model $\mathcal{S}$ and $p \in \mathcal{S}_0$. By the above lemma we get that the language recognized by $\{p\}$ is

$$L_p = L(\mathcal{A}_p) - \bigcup\{L(\mathcal{A}_q) \mid q \leq p\}$$

So given F included in $\mathcal{S}_0$, the language recognized by F is $\bigcup_{p\in F} L_p$.

For the other direction we take an automaton for every basic language in a boolean combination. We make a product of the corresponding GFP models given by Proposition 2, and take the appropriate F defined by the form of the boolean combination of the basic languages. □

Using the results in [SMGB12], it can be shown that typings in Kobayashi's type systems [Kob09c] give precisely values in GFP models.

4 A model for insightfull TAC automata

The goal of this section is to present a model capable of recognizing languages of insightful TAC automata. Theorem 7 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

In this section we build 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 elements lattice, and $\mathcal{D}_{\alpha\to\beta}$ is the set of monotone functions from $\mathcal{D}_\alpha$ 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 3. 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 [AC98] page 130).

Theorem 8 *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 this automaton accepts Ω, respectively. To capture the power of such an automaton, we are going to define a model $\mathcal{K}(Q, Q_\Omega)$ 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 9 For a given finite set Q and $Q_\Omega \subseteq Q$ we define a family of sets $\mathcal{K}_{Q,Q_\Omega} = (\mathcal{K}_\alpha)_{\alpha\in\mathcal{T}}$ that is defined by mutual recursion together with a relation $\mathcal{L} = (\mathcal{L}_\alpha)_{\alpha\in\mathcal{T}}$ 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. $\mathcal{L}_{\alpha\to\beta} = \{(f, d) \in \mathcal{K}_{\alpha\to\beta} \times \mathcal{D}_{\alpha\to\beta} \mid \forall(g, e) \in \mathcal{L}_\alpha.(f(g), d(e)) \in \mathcal{L}_\beta\}$.

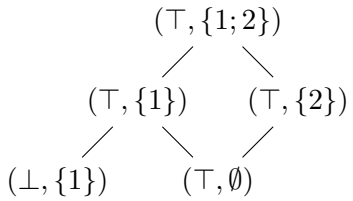

Figure 1: The order $\mathcal{K}_0$ for $Q = \{1, 2\}$ and $Q_\Omega = \{1\}$

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. 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 obtained an order preserving injection from $\mathcal{D}_0$ to $\mathcal{K}_0$.

The following lemma shows that for every type α, $\mathcal{K}_\alpha$ is a join semilattice.

Lemma 10 Given (f_1, d_1) and (f_2, d_2) in $\mathcal{L}_\alpha$, then $f_1 \vee f_2$ is in $\mathcal{K}_\alpha$ and $(f_1 \vee f_2, d_1 \vee d_2)$ is in $\mathcal{L}_\alpha$.

Proof
We proceed by induction on the structure of the type. For the base type the lemma is immediate from the definition. For the induction step consider a type of a form $\alpha \to \beta$ and two functions f_1 and f_2 in $\mathrm{mon}[\mathcal{K}_\alpha \to \mathcal{K}_\beta]$. Since, by induction, $\mathcal{K}_\beta$ is a join semilattice, we have that $f_1 \vee f_2$ is also in $\mathrm{mon}[\mathcal{K}_\alpha \to \mathcal{K}_\beta]$. By assumption of the lemma, for every (p, e) in $\mathcal{L}_\alpha$ we have that $(f_1(p), d_1(e))$ and $(f_2(p), d_2(e))$ are in $\mathcal{L}_\beta$. The induction hypothesis implies that $(f_1(p) \vee f_2(p), d_1(e) \vee d_2(e))$ is in $\mathcal{L}_\beta$. As by induction hypothesis $\mathcal{K}_\beta$ is a join semilattice, we get $(f_1 \vee f_2)(p) = f_1(p) \vee f_2(p)$ is in $\mathcal{K}_\beta$. Thus $((f_1 \vee f_2)(p), (d_1 \vee d_2)(e))$ is in $\mathcal{L}_\beta$. Since $(p, e) \in \mathcal{L}_\alpha$ was arbitrary this implies that $f_1 \vee f_2$ is in $\mathcal{K}_{\alpha\to\beta}$ and $(f_1 \vee f_2, d_1 \vee d_2)$ is in $\mathcal{L}_{\alpha\to\beta}$. □

A consequence of this lemma and of the finiteness of $\mathcal{K}_\alpha$ is that $\mathcal{K}_\alpha$ has a greatest element that we denote $\mathbb{T}_\alpha$. Observe that $(\mathbb{T}_\alpha, \top_\alpha)$ is in $\mathcal{L}_\alpha$. The lemma also implies existence of certain meets.

Corollary 11 For every type α and f_1, f_2 in $\mathcal{K}_\alpha$. If there is $g \in \mathcal{K}_\alpha$ such that $g \leq f_1$ and $g \leq f_2$ then f_1 and f_2 have a greatest lower bound $f_1 \wedge f_2$. Moreover if (f_1, d_1) and (f_2, d_2) are in $\mathcal{L}_\alpha$ then $(f_1 \wedge f_2, d_1 \wedge d_2)$ is in $\mathcal{L}_\alpha$.

Proof
Just take $f = \bigvee\{g \in \mathcal{K}_\alpha \mid g \leq f_1 \wedge g \leq f_2\}$. Element f exists by Lemma 10 and the fact that $\mathcal{K}_\alpha$ is finite. □

We are now going to show that every constant function of $\mathrm{mon}[\mathcal{K}_\alpha \to \mathcal{K}_\beta]$ is actually in $\mathcal{K}_{\alpha\to\beta}$.

Lemma 12 For every q in $\mathcal{K}_\beta$, the function c_q of $\mathrm{mon}[\mathcal{K}_\alpha \to \mathcal{K}_\beta]$ such that for every p in $\mathcal{K}_\alpha$, $c_q(p) = q$ is in $\mathcal{K}_{\alpha\to\beta}$.

Proof
To show that c_q is in $\mathcal{K}_{\alpha\to\beta}$, we need to find h_q in $\mathcal{D}_{\alpha\to\beta}$ such that for every (p,e), $(c_q(p),h_q(e))$ is in $\mathcal{L}_\beta$. Since q is in $\mathcal{K}_\beta$, there is d such that (q,d) is in $\mathcal{L}_\beta$. It suffices to take h_q to be the function of $\mathcal{D}_{\alpha\to\beta}$ such that for every e in $\mathcal{D}_\alpha$, $h_q(e)=d$. □

This lemma allows us to define inductively on types the family of constant functions $(\perp\!\!\!\perp_\alpha)_{\alpha\in\mathcal{T}}$ as follows:

1. $\perp\!\!\!\perp_0 = (\perp, Q_\Omega)$,
2. $\perp\!\!\!\perp_{\alpha\to\beta}(h) = \perp\!\!\!\perp_\beta$ for every h in $\mathcal{K}_\alpha$.

Notice that $\perp\!\!\!\perp_\alpha$ is a minimal element of $\mathcal{K}_\alpha$, but that $\mathcal{K}_\alpha$ does not have a least element in general.

For every d in $\mathcal{D}_\alpha$, we denote by L_d the set of elements of $\mathcal{K}_\alpha$ that are related to it:

$$L_d = \{p \in \mathcal{K}_\alpha \mid (p,d) \in \mathcal{L}_\alpha\}.$$

Definition 13 A type α is $\mathcal{D}$-*complete* if, for every d in $\mathcal{D}_\alpha$:

1. L_d is not empty,
2. $\perp\!\!\!\perp_\alpha \leq \bigvee L_d$,
3. for every (f,e) in $\mathcal{L}_\alpha$: $f \leq \bigvee L_d$ iff $e \leq d$.

Later we will show that every type is $\mathcal{D}$-complete, but for this we will need some preparatory lemmas.

Lemma 14 If α is a $\mathcal{D}$-complete type and d is in $\mathcal{D}_\alpha$ then $(\bigvee L_d, d)$ is in $\mathcal{L}_\alpha$.

Proof
Since α is $\mathcal{D}$-complete, L_d is not empty, and the conclusion follows directly from Lemma 10. □

Lemma 15 If α is a $\mathcal{D}$-complete type, and $d, e \in \mathcal{D}_\alpha$ then: $e \leq d$ iff $\bigvee L_e \leq \bigvee L_d$.

Proof
As α is $\mathcal{D}$-complete both L_e and L_d are not empty and therefore, $\bigvee L_e$ and $\bigvee L_d$ are well-defined. Lemma 14 also gives that $(\bigvee L_e, e)$ is in $\mathcal{L}_\alpha$. Now from $\mathcal{D}$-completeness of α, we have that $\bigvee L_e \leq \bigvee L_d$ iff $e \leq d$. □

We now define an operation $(\cdot)^{\uparrow}$ that, as we will show later, is an embedding for $\mathcal{D}$ into $\mathcal{K}$. But for this we need the notion of *co-step functions*: give two partial orders L_1 and L_2 where L_2 has a greatest element $\top$, p in L_1 and q in L_2, we write $p \nearrow q$ for the function of $\mathrm{mon}[L_1 \to L_2]$ such that for r in L_1, $(p \nearrow q)(r)$ is equal to q when $r \leq p$ and to $\top$ otherwise.

Definition 16 Let α, β be $\mathcal{D}$-complete types. For every $h \in \mathcal{D}_{\alpha\to\beta}$ and every $d \in \mathcal{D}_\alpha$ we define two monotone functions and the element $h^{\uparrow}$:

$$f_{h,d} = \bigvee L_d \nearrow \bigvee L_{h(d)}, \qquad \overline{f}_{h,d} = d \nearrow h(d),$$
$$h^{\uparrow} = \bigwedge_{d\in\mathcal{D}} f_{h,d} \ .$$

For h in $\mathcal{D}_0$, we write $h^{\uparrow}$ for $(\bot, Q_\Omega)$ when $h = \bot$ and for $(\top, Q)$ when $h = \top$.

The next lemma summarizes all essential properties of the model $\mathcal{K}$.

Lemma 17 For all $\mathcal{D}$-complete types α, β, for every $h \in \mathcal{D}_{\alpha\to\beta}$ and every $d \in \mathcal{D}_\alpha$:

1. $(f_{h,d}, \overline{f}_{h,d})$ is in $\mathcal{L}_{\alpha\to\beta}$;
2. $\perp\!\!\!\perp_{\alpha\to\beta} \leq f_{h,d}$;
3. $h^{\uparrow}$ is an element of $\mathcal{K}_{\alpha\to\beta}$ and $(h^{\uparrow}, h) \in \mathcal{L}_{\alpha\to\beta}$;
4. if $(p, e) \in \mathcal{L}_\alpha$ then $h^{\uparrow}(p) = \bigvee L_{h(e)}$;
5. $h^{\uparrow} = \bigvee L_h$.

Proof
For the first item we take $(p, e) \in \mathcal{L}_\alpha$, and show that $(f_{h,d}(p), \overline{f}_{h,d}(e)) \in \mathcal{L}_\beta$. This will be sufficient by the definition of $\mathcal{L}_{\alpha\to\beta}$. Lemma 14 gives $(\bigvee L_d, d) \in \mathcal{L}_\alpha$ and $(\bigvee L_{h(d)}, h(d)) \in \mathcal{L}_\beta$. By $\mathcal{D}$-completeness of α: $p \leq \bigvee L_d$ iff $e \leq d$. We have two cases. If $p \leq \bigvee L_d$ then $f_{h,d}(p) = \bigvee L_{h(d)}$ and $\overline{f}_{h,d}(e) = h(d)$. Otherwise, $p \not\leq \bigvee L_d$ that gives $f_{h,d}(p) = \mathbb{T}_\beta$ and $\overline{f}_{h,d}(e) = \top_\beta$. With the help of Lemma 14 in both cases we have that the result is in $\mathcal{L}_\beta$, and we are done.

For the second item, by $\mathcal{D}$-completeness of β we have $\bigvee L_{h(d)} \geq \perp\!\!\!\perp_\beta$. In the proof of the first item we have seen that $f_{h,d}(p) \geq \bigvee L_{h(d)}$ for every $p \in \mathcal{K}_\alpha$. Since $\perp\!\!\!\perp_{\alpha\to\beta}(p) = \perp\!\!\!\perp_\beta$ we get $\perp\!\!\!\perp_{\alpha\to\beta} \leq f_{h,d}$.

In order to show the third item we use the first item telling us that $(f_{h,e}, \overline{f}_{h,e})$ is in $\mathcal{L}_{\alpha\to\beta}$ for every $e \in \mathcal{D}_\alpha$. Since by the second item $\perp\!\!\!\perp_\alpha \leq f_{h,e}$, Corollary 11 shows that $(\bigwedge_{e\in\mathcal{D}_\alpha} f_{h,e}, \bigwedge_{e\in\mathcal{D}_\alpha} \overline{f}_{h,e})$ is in $\mathcal{L}_{\alpha\to\beta}$. Directly from the definition of co-step functions we have $\bigwedge_{e\in\mathcal{D}_a} e \nearrow h(e) = h$. This gives, as desired, $(\bigwedge_{e\in\mathcal{D}_\alpha} f_{h,e}, h)$ in $\mathcal{L}_{\alpha\to\beta}$.

For the fourth item, take an arbitrary $(p, e) \in \mathcal{L}_\alpha$. We show that $h^\uparrow(p) = \bigvee L_{d(e)}$. By definition $h^\uparrow(p) = \bigwedge_{e'\in\mathcal{D}_\alpha} f_{h,e'}(p)$. Moreover $f_{h,e'}(p) = \bigvee L_{h(e')}$ if $p \leq \bigvee L_{e'}$, and $f_{h,e'}(p) = \top\!\!\!\top_\beta$ otherwise. By $\mathcal{D}$-completeness of α: $p \leq \bigvee L_{e'}$ iff $e \leq e'$. So $h^\uparrow(p) = \bigwedge_{e'\in\mathcal{D}_\alpha} f_{h,e'}(p) = \bigwedge\{\bigvee L_{h(e')} : e \leq e'\}$. By Lemma 15, if $e \leq e'$ then $\bigvee L_{h(e)} \leq \bigvee L_{h(e')}$. Hence $h^\uparrow(p) = \bigvee L_{h(e)}$.

For the last item we want to show that $h^\uparrow = \bigvee L_h$. We know that $h^\uparrow \in L_h = \{g \in \mathcal{K}_{\alpha\to\beta} : (g, h) \in \mathcal{L}_\alpha\}$ since $(h^\uparrow, h) \in \mathcal{L}_{\alpha\to\beta}$ by the third item. We show that for every $g \in L_h$, $g \leq h^\uparrow$. Take some $(p, e) \in \mathcal{L}_\alpha$. We have $(g(p), h(e)) \in \mathcal{L}_\beta$, hence $g(p) \leq \bigvee L_{h(e)}$ by definition of $L_{h(e)}$. Since $h^\uparrow(p) = \bigvee L_{h(e)}$ by the fourth item, we get $g \leq h^\uparrow$. □

Lemma 18 Every type α is $\mathcal{D}$-complete.

Proof
This is proved by induction on the structure of the type. The case of the base type follows by direct examination. For the induction step consider a type $\alpha \to \beta$ and suppose that α and β are $\mathcal{D}$-complete. Given d in $\mathcal{D}_{\alpha\to\beta}$, Lemma 17 gives that $(d^\uparrow, d)$ is in $\mathcal{L}_{\alpha\to\beta}$ proving that $L_d \neq \emptyset$, it also gives that $\perp\!\!\!\perp_{\alpha\to\beta} \leq d^\uparrow$ and $d^\uparrow = \bigvee L_d$ so that we obtain $\perp\!\!\!\perp_{\alpha\to\beta} \leq \bigvee L_d$. It just remains to prove that given (f, e) in $\mathcal{L}_{\alpha\to\beta}$, $f \leq \bigvee L_d$ iff $e \leq d$.

Let's first suppose that $e \leq d$. Take a $p \in \mathcal{K}_\alpha$. By definition of the model there is e', such that $(p, e') \in \mathcal{L}_\alpha$. Lemma 15 gives us $\bigvee L_{e(e')} \leq \bigvee L_{d(e')}$. By definition of $\mathcal{L}_{\alpha\to\beta}$ we have that $(f(p), e(e')) \in \mathcal{L}_\beta$, so $f(p) \leq \bigvee L_{e(e')}$ by definition of $L_{e(e')}$. This gives $f(p) \leq \bigvee L_{e(e')} \leq \bigvee L_{d(e')}$. On the other hand Lemma 17 gives that $\bigvee L_{d(e')} = d^\uparrow(p) = (\bigvee L_d)(p)$. This shows the desired $f(p) \leq (\bigvee L_d)(p)$ for every $p \in \mathcal{K}_\alpha$.

Let us now suppose that $f \leq \bigvee L_d$. The $\mathcal{D}$-completeness of α tells us that for every e' in $\mathcal{D}_\alpha$ there is p in $\mathcal{K}_\alpha$ so that (p, e') is in $\mathcal{L}_\alpha$. Then Lemma 17 gives that $(\bigvee L_d)(p) = \bigvee L_{d(e')}$. We have thus $f(p) \leq (\bigvee L_d)(p) = \bigvee L_{d(e')}$. Using Lemma 15 and the fact that $(f(p), e(e')) \in \mathcal{L}_\beta$ we get $e(e') \leq d(e')$. As e' was arbitrary we obtain that $e \leq d$. □

The proposition below sums up the properties of the embedding $(\cdot)^\uparrow$ from Definition 16.

Proposition 3 *Given a type α, and d in $\mathcal{D}_\alpha$, there is $d^\uparrow$ in $\mathcal{K}_\alpha$ such that:*

1. *$(d^\uparrow, d)$ is in $\mathcal{L}_\alpha$,*
2. *if $e \in \mathcal{D}_\alpha$ and $d \leq e$ then $d^\uparrow \leq e^\uparrow$,*
3. *if (f, d) is in $\mathcal{L}_\alpha$, then $f \leq d^\uparrow$,*
4. *if $\alpha = \alpha_1 \to \alpha_2$ and (g, e) is in $\mathcal{L}_{\alpha_1}$ then $d^\uparrow(g) = (d(e))^\uparrow$*

Proof
These properties follow directly from Lemma 17, except for the second property for which a small calculation is needed. Since $(d^\uparrow, d)$ is in $\mathcal{L}_\alpha$ and $d \leq e$ then by Lemma 17: $d^\uparrow \leq \bigvee L_e$. The later is precisely $e^\uparrow$ by Lemma 17. □

The next lemma shows that the relation $\mathcal{L}_\alpha$ is functional.

Lemma 19 For every type α and f in $\mathcal{K}_\alpha$: if (f, d_1) and (f, d_2) are in $\mathcal{L}_\alpha$, then $d_1 = d_2$.

Proof
We proceed by induction on the structure of the type. The case of the base type follows from a direct inspection. For the induction step suppose that both (f, d_1) and (f, d_2) are in $\mathcal{L}_{\alpha\to\beta}$. Take arbitrary $e \in \mathcal{D}_\alpha$. By Lemma 17 we have $(e^\uparrow, e) \in \mathcal{L}_\alpha$. Therefore $(f(e^\uparrow), d_1(e))$ and $(f(e^\uparrow), d_2(e))$ in $\mathcal{L}_\beta$. The induction hypothesis implies that $d_1(e) = d_2(e)$. Since e was arbitrary we get $d_1 = d_2$. □

Since, by definition, for every $f \in \mathcal{K}_\alpha$ we have $(f, d) \in \mathcal{L}_\alpha$ for some $d \in \mathcal{D}_\alpha$, the above lemma gives us a projection of $\mathcal{K}_\alpha$ to $\mathcal{D}_\alpha$.

Definition 20 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 immediately state some properties of the projection.

Lemma 21 Given f in $\mathcal{K}_{\alpha\to\beta}$ and p in $\mathcal{K}_\alpha$, $\overline{f(p)} = \overline{f}(\overline{p})$.

Proof
We have $(f, \overline{f})$ in $\mathcal{L}_{\alpha\to\beta}$ and $(p, \overline{p})$ in $\mathcal{L}_\alpha$, so that $(f(p), \overline{f}(\overline{p}))$ is in $\mathcal{L}_\beta$ and thus $\overline{f(p)} = \overline{f}(\overline{p})$. □

Lemma 22 Given f and g in $\mathcal{K}_\alpha$, if $f \leq g$ then $\overline{f} \leq \overline{g}$.

Proof
We proceed by induction on the structure of the types. The case of the base type follows by a straightforward inspection. For the induction step take $f \leq g$ in $\mathcal{K}_{\alpha\to\beta}$. For an arbitrary $d \in \mathcal{D}_\alpha$ we have $f(d^\uparrow) \leq g(d^\uparrow)$. By induction hypothesis on type β we get $\overline{f(d^\uparrow)} \leq \overline{g(d^\uparrow)}$. By Lemma 21 we obtain $\overline{f(d^\uparrow)} = \overline{f}(\overline{d^\uparrow}) = \overline{f}(d)$. The last equality follows from the fact that $\overline{d^\uparrow} = d$ since $(d^\uparrow, d)$ is in $\mathcal{L}_\alpha$ by Proposition 3. Of course the same equalities hold for g too. So $\overline{f}(d) \leq \overline{g}(d)$ for arbitrary d, and we are done. □

We are now going to give the definition of the interpretation of the fixpoint combinator in $\mathcal{K}$. This definition is based on that of the fixpoint operator in $\mathcal{D}$. As a shorthand, we write fix_α for the operation in $\mathcal{D}_{(\alpha\to\alpha)\to\alpha}$ that maps a function of $\mathcal{D}_{\alpha\to\alpha}$ to its least fixpoint.

Lemma 23 Given f in $\mathcal{K}_{\alpha\to\alpha}$, we have $f(\mathrm{fix}_\alpha(\overline{f})^\uparrow) \leq \mathrm{fix}_\alpha(\overline{f})^\uparrow$.

Proof
By Proposition 3, we have that $(\mathrm{fix}_\alpha(\overline{f})^\uparrow, \mathrm{fix}_\alpha(\overline{f}))$ is in $\mathcal{L}_\alpha$. Moreover, $(f, \overline{f})$ is in $\mathcal{L}_{\alpha\to\alpha}$, therefore, we have $(f(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \overline{f}(\mathrm{fix}_\alpha(\overline{f}))) = (f(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \mathrm{fix}_\alpha(\overline{f}))$ is in $\mathcal{L}_\alpha$. Then by Proposition 3 we have $f(\mathrm{fix}_\alpha(\overline{f})^\uparrow) \leq \mathrm{fix}_\alpha(\overline{f})^\uparrow$. □

The above lemma guarantees that the sequence $f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow)$ is decreasing. We can now define an operator that, as we will show, is the fixpoint operator we are looking for.

Definition 24 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 show that Fix_α is monotone.

Lemma 25 Given f and g in $\mathcal{K}_{\alpha\to\alpha}$, if $f \leq g$ then $\mathrm{Fix}_\alpha(f) \leq \mathrm{Fix}_\alpha(g)$.

Proof
By Lemma 22, $f \leq g$ implies $\overline{f} \leq \overline{g}$, as fix_α is monotone, we have $\mathrm{fix}_\alpha(\overline{f}) \leq \mathrm{fix}_\alpha(\overline{g})$ and $\mathrm{fix}_\alpha(\overline{f})^\uparrow \leq \mathrm{fix}_\alpha(\overline{g})^\uparrow$ by Proposition 3. As $f \leq g$ we have $f^k(\mathrm{fix}_\alpha(\overline{f})^\uparrow) \leq g^k(\mathrm{fix}_\alpha(\overline{g})^\uparrow)$ for every k in $\mathbb{N}$. Therefore $\bigwedge_{n\in\mathbb{N}} f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow) \leq \bigwedge_{n\in\mathbb{N}} g^n(\mathrm{fix}_\alpha(\overline{g})^\uparrow)$. □

The last step is to show that Fix_α is actually in $\mathcal{K}_{(\alpha\to\alpha)\to\alpha}$.

Lemma 26 For every α, Fix_α is in $\mathcal{K}_\alpha$ and $(\mathrm{Fix}_\alpha, \mathrm{fix}_\alpha)$ is in $\mathcal{L}_{(\alpha\to\alpha)\to\alpha}$.

Proof
We know that $(f, \overline{f})$ in $\mathcal{L}_{\alpha\to\alpha}$. By Proposition 3, we have $(\mathrm{fix}_\alpha(\overline{f})^\uparrow, \mathrm{fix}_\alpha(\overline{f}))$ in $\mathcal{L}_\alpha$. So $(f(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \overline{f}(\mathrm{fix}_\alpha(\overline{f}))) = (f(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \mathrm{fix}_\alpha(\overline{f}))$ is in $\mathcal{L}_\alpha$. Repeating this argument we have that for every $n \in \mathbb{N}$, $(f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \mathrm{fix}_\alpha(\overline{f}))$ is in $\mathcal{L}_\alpha$. But $f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow)$ is decreasing by Lemma 23. Since $\mathcal{K}_\alpha$ is finite, we get $(\bigwedge_{n\in\mathbb{N}} f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow), \mathrm{fix}_\alpha(\overline{f}))$ in $\mathcal{L}_\alpha$. We are done since $\bigwedge_{n\in\mathbb{N}} f^n(\mathrm{fix}_\alpha(\overline{f})^\uparrow) = \mathrm{Fix}_\alpha(f)$. □

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

Definition 27 For a finite set Q and $Q_\Omega \subseteq Q$ consider a tuple $\mathcal{K}(Q, Q_\Omega, \rho) = (\{\mathcal{K}_\alpha\}_{\alpha\in\mathcal{T}}, \rho)$ where $\{\mathcal{K}_\alpha\}_{\alpha\in\mathcal{T}}$ is as in Definition 9 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 $(\bot, Q_\Omega)$.

We will show $\mathcal{K}(Q, Q_\Omega, \rho)$ is indeed a model of λY-calculus. Since $\mathcal{K}_{\alpha\to\beta}$ does not contain all the functions from $\mathcal{K}_\alpha$ to $\mathcal{K}_\beta$ we must show that there are enough of them to form a model of λY, the main problem being to show that $[\![\lambda x.M]\!]^v_{\mathcal{K}}$ defines an element of $\mathcal{K}$. For this it will be more appropriate to consider the semantics of a term as a function of values of its free variables. Given a finite sequence of variables $\vec{x} = x_1, \ldots, x_n$ of types $\alpha_1, \ldots, \alpha_n$ respectively and a term M of type β with free variables in $\vec{x}$, the meaning of M in the model $\mathcal{K}$ with respect to $\vec{x}$ will be a function $[\![M]\!]_{\vec{x},\mathcal{K}}$ in $\mathcal{K}_{\alpha_1} \to \cdots \to \mathcal{K}_{\alpha_n} \to \mathcal{K}_\beta$ that represents the function $\lambda\vec{p}.[\![M]\!]^{[p_1/x_1,\ldots,p_n/x_n]}_{\mathcal{K}}$. Formally it is defined as follows:

1. $[\![Y^{(\beta\to\beta)\to\beta}]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\ \mathrm{Fix}_\beta$
2. $[\![a]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\ \rho(a)$
3. $[\![x_i^{\alpha_i}]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.p_i$
4. $[\![\omega^\beta]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\ \top_\beta$
5. $[\![\Omega^\beta]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\ \bot\!\!\!\bot_\beta$
6. $[\![MN]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\ ([\![M]\!]_{\vec{x},\mathcal{K}}\,\vec{p})([\![N]\!]_{\vec{x},\mathcal{K}}\,\vec{p})$
7. $[\![\lambda y.M]\!]_{\vec{x},\mathcal{K}} = \lambda\vec{p}.\lambda p_y.\ [\![M]\!]_{\vec{x}y,\mathcal{K}}\vec{p}p_y$

Note that λ symbol on the right hand side of the equality is the semantic symbol used to denote a relevant function, and not a part of the syntax while the sequence $\vec{p}$ denote a sequence of elements $p_1, \ldots, p_n$ respectively in $\mathcal{K}_{\alpha_1}, \ldots, \mathcal{K}_{\alpha_n}$.

Lemma 26 ensures the existence of the meaning of Y in $\mathcal{K}$. With this at hand, the next lemma provides all the other facts necessary to show that the meaning of a term with respect to $\vec{x}$ is always an element of the model.

Lemma 28 For every sequence of types $\vec{\alpha} = \alpha_1 \dots \alpha_n$ and every types β, γ we have the following:

- For every constant $p \in \mathcal{K}_\beta$ the constant function $f_p : \alpha_1 \to \cdots \to \alpha_n \to \beta$ belongs to $\mathcal{K}$.
- For $i = 1, \dots, n$, the projection $\pi_i : \alpha_1 \to \cdots \to \alpha_n \to \alpha_i$ belongs to $\mathcal{K}$.
- If $f : \vec{\alpha} \to (\beta \to \gamma)$ and $g : \vec{\alpha} \to \beta$ are in $\mathcal{K}$ then $\lambda\vec{p}.f\vec{p}(g\vec{p}) : \vec{\alpha} \to \gamma$ is in $\mathcal{K}$.

Proof
The first item of the Lemma is given by Lemma 12, the second does not present more difficulty. Finally the third proceeds by a direct examination once we observe the following property of $\mathcal{K}(Q, Q_\Omega, \rho)$. Given two elements f of $\mathrm{mon}[\mathcal{K}_{\alpha_1} \to \cdots \to \mathrm{mon}[\mathcal{K}_{\alpha_n} \to \mathcal{K}_\beta]]$ and g of $\mathcal{D}_{\alpha_1 \to \cdots \to \alpha_n \to \beta}$, if for every $d_1, \dots, d_n$ in $\mathcal{K}_{\alpha_1}, \dots, \mathcal{K}_{\alpha_n}$, $(f(d_1, \dots, d_n), g(\overline{d_1}, \dots, \overline{d_n})) \in \mathcal{L}_\beta$ then f is in $\mathcal{K}_{\alpha_1 \to \cdots \to \alpha_n \to \beta}$ and (f, g) is in $\mathcal{L}_{\alpha_1 \to \cdots \to \alpha_n \to \beta}$. This observation follows directly from Proposition 3 and the definition of the model. □

Theorem 29 *For every finite set Q and every set $Q_\Omega \subseteq Q$ the model $\mathcal{K}(Q, Q_\Omega, \rho)$ as in Definition 27 is a model of the λY-calculus.*

Let us mention the following useful fact that showing a correspondence between the meanings of a term in $\mathcal{K}$ and in $\mathcal{D}$. The proof is immediate since $\{\mathcal{L}_\alpha\}_{\alpha \in \mathcal{T}}$ is a logical relation (cf [AC98]).

Lemma 30 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 is indeed sufficient to recognize languages of TAC automata. For the rest of the section fix a tree signature Σ and a TAC automaton

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta_1 : Q \times \Sigma_1 \to \{ff, tt\}, \delta_2 : Q \times \Sigma_2 \to \mathcal{P}(Q^2) \rangle \,.$$

Consider a model $\mathcal{K}$ based on $\mathcal{K}(Q, Q_\Omega, \rho)$ as in Definition 27, where Q_Ω is the set of states q such that $\delta(q, \Omega) = tt$. It remains to specify the meaning of constants like $c : 0$ or $a : 0^2 \to 0$ in Σ:

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

Lemma 31 For every a in Σ of type $o^2 \to o$: $\rho(a)$ is in $\mathcal{K}_{o^2 \to o}$ and $(\rho(a), \top_{o^2 \to o})$ is in $\mathcal{L}_{o^2 \to o}$.

Proof
It is easy to see that $\rho(a)$ is monotone. For the membership in $\mathcal{K}$ the witnessing function from $\mathcal{D}_{o^2 \to o}$ is $\top_{0^2 \to 0}$. □

Once we know that $\mathcal{K}$ is a model we can state some of its useful properties. The first one tells what the meaning of unsolvable terms is. The second indicates how unsolvability is taken into account in the computation of a fixpoint.

Proposition 4 *Given a closed term M of type 0: $BT(M) = \Omega^0$ iff $[\![M]\!]_{\mathcal{K}} = (\bot, Q_\Omega)$.*

Proof
In case $[\![M]\!]_{\mathcal{K}} = (\bot, Q_\Omega)$, Lemma 30 gives us $[\![M]\!]_{\mathcal{D}} = \bot$. By Theorem 8 this implies $BT(M) = \Omega^0$.

In case $BT(M) = \Omega^0$, Theorem 8 entails that $[\![M]\!]_{\mathcal{D}} = \bot$. By Lemma 30 $([\![M]\!]_{\mathcal{K}}, \bot)$ is in $\mathcal{L}_0$. But this is possible only if $[\![M]\!]_{\mathcal{K}} = (\bot, Q_\Omega)$. □

Lemma 32 Given a type $\beta = \beta_1 \to \cdots \to \beta_l \to 0$, a sequence of types $\vec{\alpha} = \alpha_1, \ldots, \alpha_k$, and a function $f \in \mathcal{K}_{\vec{\alpha} \to \beta \to \beta}$, consider functions:

$$h = \lambda p_1 \ldots p_k. \left(\mathrm{fix}_\alpha(\overline{f(p_1) \ldots (p_k)})\right)^\uparrow \qquad g = \lambda e_1 \ldots e_k. \mathrm{fix}_\alpha(\overline{f}(e_1) \ldots (e_k))$$

that are respectively in $\mathcal{K}_{\beta_1} \to \cdots \to \mathcal{K}_{\beta_k} \to \mathcal{K}_\alpha$ and in $\mathcal{D}_{\vec{\beta} \to \alpha}$. Then h is in $\mathcal{K}_{\vec{\beta} \to \alpha}$ and (h, g) is in $\mathcal{L}_{\vec{\beta} \to \alpha}$. Moreover, for every $p_1 \in \mathcal{K}_{\beta_1}$, …, $p_k \in \mathcal{K}_{\beta_k}$, $q_1 \in \mathcal{K}_{\alpha_1}$,…, $q_l \in \mathcal{K}_{\alpha_l}$ we have

$$h(p_1, \ldots, p_k)(q_1, \ldots, p_l) = \begin{cases} (\bot, Q_\Omega) & \text{if } g(\overline{p}_1, \ldots, \overline{p}_k)(\overline{q}_1, \ldots, \overline{q}_l) = \bot \\ (\top, Q) & \text{if } g(\overline{p}_1, \ldots, \overline{p}_k)(\overline{q}_1, \ldots \overline{q}_l) = \top \end{cases}$$

Proof
To prove that (h, g) is in $\mathcal{L}_{\vec{\alpha}\to\beta}$, we resort to the remark we made in the proof of Lemma 28, so that it suffices to show that for every p_1, ..., p_k respectively in $\mathcal{K}_{\alpha_1}$, ..., $\mathcal{K}_{\alpha_k}$, $(h(p_1,\ldots,p_k), g(\overline{p}_1,\ldots,\overline{p}_k))$ is in $\mathcal{L}_\beta$. We have that $h(p_1,\ldots,p_k) = \left(\mathrm{fix}_\alpha(\overline{f(p_1,\ldots,p_k)})\right)^\uparrow$ that is in $\mathcal{K}_\alpha$, and we have that

$$\begin{array}{rcl}
\overline{h(p_1,\ldots,p_k)} & = & \overline{\left(\mathrm{fix}_\alpha(\overline{f(p_1,\ldots,p_k)})\right)^\uparrow} \\
& = & \mathrm{fix}_\alpha(\overline{f(p_1,\ldots,p_k)}) \\
& = & \mathrm{fix}_\alpha(\overline{f}(\overline{p}_1,\ldots,\overline{p}_k)) \text{ by successive use of Lemma 21} \\
& = & g(\overline{p_1},\ldots,\overline{p}_k)
\end{array}$$

This shows that (h, g) is in $\mathcal{L}_{\vec{\alpha}\to\beta}$ and thus h is in $\mathcal{K}_{\vec{\alpha}\to\beta}$.

Given r in $\mathcal{D}_{\gamma\to\delta}$, from the fourth item of Proposition 3, we have that whenever (q, e) is in $\mathcal{L}_\alpha$, then $r^\uparrow(q) = (r(e))^\uparrow$, so that in particular $r^\uparrow(q) = (r(\overline{q}))^\uparrow$. A simple induction shows then that, for r in $\mathcal{D}_{\gamma_1\to\cdots\to\gamma_n\to\delta}$,

$$r^\uparrow(q_1,\ldots,q_n) = (r(\overline{q_1},\ldots,\overline{q_n}))^\uparrow$$

Therefore if $\delta = 0$ and $r(\overline{q_1},\ldots,\overline{q_n}) = \bot$, we have $(r(\overline{q_1},\ldots,\overline{q_n}))^\uparrow = (\bot, Q_\Omega)$. Moreover, in case $r(\overline{q_1},\ldots,\overline{q_n}) = \top$, we have $(r(\overline{q_1},\ldots,\overline{q_n}))^\uparrow = (\top, Q)$. The lemma follows from choosing $r = g(\overline{p}_1,\ldots,\overline{p}_k)$ and remarking that $(g(\overline{p}_1,\ldots,\overline{p}_k))^\uparrow = h(p_1,\ldots,p_k)$. □

As in the case of GFP-models the semantics of a Böhm tree is defined in terms of its truncations: $[\![BT(M)]\!]_\mathcal{K} = \bigwedge\{[\![BT(M)\downarrow_n]\!]_\mathcal{K} : n \in \mathbb{N}\}$. The subtle 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 33 *For very closed term M of type 0: $[\![M]\!]_\mathcal{K} = [\![BT(M)]\!]_\mathcal{K}$.*

Proof
First we show that $[\![M]\!]_\mathcal{K} \leq [\![BT(M)]\!]_\mathcal{K}$. For this we define a finite approximation of the Böhm tree. *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 it by induction:

- $ABT_0(M) = M$;

- $ABT_{l+1}(M)$ is M if M does not have head normal form otherwise it is a term $\lambda x.N_0 ABT_l(N_1)\dots ABT_l(N_k)$, where $\lambda x.N_0 N_1 \dots N_k$ is the head normal form of M.

Since $ABT_l(M)$ is obtained from M by a sequence of $\beta\delta$-reductions, $[\![M]\!]_{\mathcal{K}} = [\![ABT_l(M)]\!]_{\mathcal{K}}$ for every l. We now show that for every term M and every l:

$$[\![ABT_l(M)]\!]_{\mathcal{K}} \leq [\![BT(M)\downarrow_l]\!]_{\mathcal{K}}.$$

Up to depth l, the two terms have the same structure as trees. We will see 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 a term without head normal form and on the other Ω^0. By Proposition 4 the two have the same meaning in $\mathcal{S}$.

The desired inequality $[\![M]\!]_{\mathcal{K}} \leq [\![BT(M)]\!]_{\mathcal{K}}$ follows now directly from the definition of the semantics of $BT(M)$ since $[\![M]\!]_{\mathcal{K}} = [\![ABT_l(M)]\!]_{\mathcal{K}} \leq [\![BT(M)\downarrow_l]\!]_{\mathcal{K}}$ for every $l \in \mathbb{N}$.

For the inequality in the other direction we will also introduce a new notion. 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. For this we introduce new constants c_N for every subterm YN of M. The type of c_N is $\vec{\alpha} \to \beta$ if β is the type of YN and $\vec{\alpha} = \alpha_1 \dots \alpha_k$ is the sequence of types of the sequence of free variables $\vec{x} = x_1 \dots x_k$ occurring in YN. We let the semantics of a constant c_N be

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

First we need to check that indeed $[\![c_N]\!]$ is in $\mathcal{K}$. For this we have prepared Lemma 32. Indeed $[\![c_N]\!]_{\mathcal{K}} = \lambda p_1 \dots p_k.\ \left(\mathrm{fix}_\beta(\overline{f(p_1,\dots,p_k)})\right)^\uparrow$, for $f = \lambda\vec{p}.\ [\![N]\!]^{[\vec{p}/\vec{x}]}$. So $[\![c_N]\!]_{\mathcal{K}}$ is h from Lemma 32 and $[\![c_N]\!]_{\mathcal{D}} = \overline{[\![c_N]\!]_{\mathcal{K}}}$ is g from that lemma. The lemma additionally gives us that 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 $n \in \mathbb{N}$.

$$\begin{aligned}\mathit{iterate}^0(N) =& c_N\\ \mathit{iterate}^{n+1}(N) =& \lambda\vec{x}.N(\mathit{iterate}^n(N)\vec{x})\ .\end{aligned}$$

From the definition of the fixpoint operator in $\mathcal{K}$ and the fact that $\mathcal{K}_\beta$ is finite it follows that $[\![\mathit{iterate}^n(N)]\!] = [\![\lambda\vec{x}.YN]\!]$ 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 $\mathit{iterate}^i(N)\vec{x}$ 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. To finish the proof 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 occurences of Y. For a term K let $\tilde{K}$ stand for a term obtained from K by simultaneously replacing c_N by $\lambda\vec{x}.YN$. Because of Lemma 23, we have $[\![c_N]\!]_\mathcal{K} \geq [\![\lambda\vec{x}.YN]\!]_\mathcal{K}$ 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 $(\lambda\vec{x}.YN)\tilde{K}_1 \dots \tilde{K}_i$. Observe that both terms are closed and of type 0. We will be done if we show that $[\![c_N K_1 \dots K_i]\!]_\mathcal{K} \geq [\![BT((\lambda\vec{x}.YN)\tilde{K}_1 \dots \tilde{K}_i)]\!]_\mathcal{K}$.

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, $[\![c_N]\!]_\mathcal{K} \geq [\![\lambda\vec{x}.\ YN]\!]_\mathcal{K}$ we get $[\![YN\tilde{K}_1 \dots \tilde{K}_i]\!]_\mathcal{K} = (\bot, Q_\Omega)$. But then Proposition 4 implies that $YNK_1 \dots K_i$ is unsolvable. Thus $[\![BT((\lambda\vec{x}NY)\tilde{K}_1 \dots \tilde{K}_i)]\!]_\mathcal{K} = [\![\Omega]\!]_\mathcal{K} = (\bot, Q_\Omega)$.

□

Theorem 34 *Let $\mathcal{A}$ be an insightful TAC automaton and $\mathcal{K}$ a model as constructed at the beginning of the subsection. For every closed term M of type 0:*

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

Proof
Recall that $\mathcal{K}$ is constructed over the sets Q and Q_Ω where Q is the set of states of $\mathcal{A}$ and Q_Ω is the set of states that from which $\mathcal{A}$ accepts Ω: $Q_\Omega = \{q \mid \delta(q, \Omega) = tt\}$.

For the left to right implication suppose that $\mathcal{A}$ accepts $BT(M)$. Since, by Theorem 33, $[\![M]\!] = [\![BT(M)]\!]$ it is enough to show that q^0, that is the initial state of $\mathcal{A}$, is in the second component of $[\![BT(M)]\!]$. For this we show that q^0 is in the second component of $[\![BT(M)\!\downarrow_l]\!]$ for every $l \in M$.

The tree $BT(M)$ is a ranked tree labeled with constants form the signature. The run of $\mathcal{A}$ is function r assigning to every node a state of $\mathcal{A}$. The tree $BT(M)\!\downarrow_l$ is a prefix of this tree containing nodes up to depth l. Let us call it t_l. Every node v in the domain of t_l corresponds to a subterm of $BT(M)\!\downarrow_l$ that we denote M^l_v.

By induction on the height of v we show that $r(v)$ appears in the second component of $[\![M^l_v]\!]$. If v is a leaf at depth l then M_v is ω^0. We are done since $[\![\omega^0]\!] = (\top, Q)$. If v is a leaf of depth smaller than l then M^l_v is Ω^0 or a constant c of type 0. In the later case by definition of a run, we have $r(v) \in \{q \mid \delta(q, c) = tt\}$. We are done by the semantics of c in the model. If M^l_v is Ω^0 then $[\![M^l_v]\!] = (\bot, Q_\Omega)$ and $r(v)$ belongs to Q_Ω by definition of the run. The last case is when v is an internal node of the tree t_l. In this case $M^l_w = aM^l_{v1}M^l_{v2}$ where a is the constant labeling v in t_l. By induction assumption we have that $r(vi)$ appears in the second component of $[\![M^l_{vi}]\!]$, and we are done by using the semantics of a.

For the direction from right to left we suppose that q^0 in the second component of $[\![M]\!]$. By Theorem 33, $[\![M]\!] = [\![BT(M)]\!]$. We will construct a run of $\mathcal{A}$ on $BT(M)$.

If M does not have head normal form then $[\![M]\!] = (\bot, Q_\Omega)$ by Proposition 4. In this case $BT(M)$ is the tree consisting only of the root labeled Ω^0. Hence $q^0 \in Q_\Omega$ and we are done.

Otherwise $BT(M)$ has some letter a in the root. In case it is a leaf, the conclusion is immediate. In case it is a binary symbol, since q^0 is in the second component of $[\![BT(M)]\!]$, it is also in the second component of $[\![BT(M)\!\downarrow_l]\!]$ for all l. We know that $BT(M)\!\downarrow_l$ is of the form $aN^l_1N^l_2$ for some N^l_1, N^l_2. As $\delta_{\mathcal{A}}(q^0, a)$ is a finite set of pairs, there is a pair $(q_1, q_2) \in \delta_{\mathcal{A}}(q^0, a)$

such that q_1 is in the second component of $[\![M_1^l]\!]$ and q_2 is in the second component of $[\![M_2^l]\!]$ for infinitely many l. We put $r(1) = q_1$ and $r(2) = q_2$ and repeat the argument starting from the nodes 1 and 2 respectively. It is easy to see that this inductive procedure gives a, potentially infinite, run of $\mathcal{A}$. Hence $BT(M) \in L(\mathcal{A})$ as every run of $\mathcal{A}$ is accepting. $\square$

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 need also 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 35 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 36 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 [SW12]).

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 $M_1, \ldots M_n$ which 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 since we may represent the term d_i of type $[\alpha]$ with the i^{th} projection $\lambda x_1 \ldots x_n.x_i$ by letting $[\alpha] = \beta^k \to \beta$. When M represents d_i, *i.e.* is equal to $\lambda x_1 \ldots x_n.x_i$, the term $MM_1 \ldots M_k$ reduces to M_i and thus represents well the semantic of the *case* construct.

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

$$\begin{aligned} \alpha^\bullet &= \alpha \text{ when } \alpha \text{ is atomic} \\ (\alpha \to \beta)^\bullet &= \alpha^\bullet \to [\alpha] \to \beta^\bullet \end{aligned}$$

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 $\mathcal{S}$.

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

To prove correctness of this translation, we need two lemmas.

Lemma 37 Given a term M and a valuation υ, and the terms $N_1, \ldots, N_n$ we have the following identity:

$$[M.\sigma, \upsilon] = \left[M, \upsilon'\right].\sigma'$$

where $\sigma = [N_1/x_1^{\alpha_1}, \ldots, N_n/x_n^{\alpha_n}]$, $\sigma' = [[N_1, \upsilon]/x_1^{\alpha_1^\bullet}, \ldots, [N_n, \upsilon]/x_n^{\alpha_n^\bullet}]$ and $\upsilon' = \upsilon[[\![N_1]\!]^\upsilon/x_1^{\alpha_1}, \ldots, [\![N_n]\!]^\upsilon/x_n^{\alpha_n}]$.

Proof
We proceed by induction on the structure of M.

In case M is a variable different from the variables $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}$ or is a constant, then the result is obvious. In case $M = x_i^{\alpha_i}$, the conclusion also follows with no difficulty.

In case $M = M_1 M_2$ then, $[M.\sigma, \upsilon] = [M_1.\sigma, \upsilon]\,[M_2.\sigma, \upsilon]\,[\![M_2.\sigma]\!]^\upsilon$, but, by induction, $[M_1.\sigma, \upsilon] = [M_1, \upsilon'].\sigma'$, $[M_2.\sigma, \upsilon] = [M_2, \upsilon'].\sigma'$; moreover, $[\![M_2.\sigma]\!]^\upsilon = [\![M_2]\!]^{\upsilon'}$. Now we have that

$$\begin{array}{rcl}
\left[M, \upsilon'\right].\sigma' & = & \left[M_1, \upsilon'\right].\sigma' \left[M_2, \upsilon'\right].\sigma' [\![M_2]\!]^{\upsilon'} \\
& = & [M_1.\sigma, \upsilon]\,[M_2.\sigma, \upsilon]\,[\![M_2.\sigma]\!]^\upsilon \\
& = & [M.\sigma, \upsilon]
\end{array}$$

In case $M = \lambda x^\alpha.N$ (we assume that x^α is different from the $x_i^{\alpha_i}$), then $[\lambda x^\alpha.M.\sigma, \upsilon] = \lambda x^{\alpha^\bullet} y^{[\alpha]}.\text{case } y^{[\alpha]}\{f \to M.\sigma, \upsilon[f/x^\alpha]\}_{f \in \mathcal{S}_\alpha}$. By induction we have that, for every f in $\mathcal{M}_\alpha$ $[M.\sigma, \upsilon[f/x^\alpha]] = [M, \upsilon'[f/x^\alpha]].\sigma'$. But,

$$\begin{array}{rcl}
\left[\lambda x^\alpha.M, \upsilon'\right].\sigma' & = & (\lambda x^{\alpha^\bullet} y^{[\alpha]}.\text{case } y^{[\alpha]}\{f \to \left[M, \upsilon'[f/x^\alpha]\right]\}_{f \in \mathcal{S}_\alpha}).\sigma' \\
& = & \lambda x^{\alpha^\bullet} y^{[\alpha]}.\text{case } y^{[\alpha]}\{f \to \left[M, \upsilon'[f/x^\alpha]\right].\sigma'\}_{f \in \mathcal{S}_\alpha} \\
& = & \lambda x^{\alpha^\bullet} y^{[\alpha]}.\text{case } y^{[\alpha]}\{f \to [M.\sigma, \upsilon[f/x^\alpha]]\}_{f \in \mathcal{S}_\alpha} \\
& = & [\lambda x^\alpha.M.\sigma, \upsilon]
\end{array}$$

In case $M = YN$, then $[M.\sigma, \upsilon] = Y(\lambda y^{\alpha^\bullet}.\,[N.\sigma, \upsilon]\, y^{\alpha^\bullet} [\![M]\!]^\upsilon)$. By induction, we have that $[M.\sigma, \upsilon] = [M, \upsilon'].\sigma'$; furthermore, $[\![M]\!]^{\upsilon'} = [\![M.\sigma]\!]^\upsilon$ so that,

$$\begin{array}{rcl}
\left[M, \upsilon'\right].\sigma' & = & Y(\lambda y^{\alpha^\bullet}.\left[N, \upsilon'\right] y^{\alpha^\bullet} [\![M]\!]^{\upsilon'}).\sigma' \\
& = & Y(\lambda y^{\alpha^\bullet}.\left[N, \upsilon'\right].\sigma' y^{\alpha^\bullet} [\![M]\!]^{\upsilon'}) \\
& = & Y(\lambda y^{\alpha^\bullet}.\,[N.\sigma, \upsilon]\, y^{\alpha^\bullet} [\![M.\sigma]\!]^\upsilon) \\
& = & [M.\sigma, \upsilon]
\end{array}$$

□

Lemma 38 If $M \to_{\beta\delta h} M'$, then $[M, \upsilon] \to^{+}_{\beta\delta h} [M', \upsilon]$.

Proof
We proceed by induction on the structure of M. We only treat the cases where M is a redex, the other cases being trivial by induction. We are left with two cases: $M = (\lambda x^{\alpha}.P)Q$ and $M = Y^{(\alpha\to\alpha)\to\alpha}P$.

In case $M = (\lambda x^{\alpha}.P)Q$, we have that $M' = P[Q/x^{\alpha}]$, and using the Lemma 37 we have that $[M', \upsilon] = [P, \upsilon[[\![Q]\!]^{\upsilon}/x^{\alpha}]]\,[[Q, \upsilon]/x^{\alpha}]$. But then we have

$$\begin{array}{rcl}
[M, \upsilon] & = & [\lambda x^{\alpha}.P, \upsilon]\,[Q, \upsilon]\,[\![Q]\!]^{\upsilon} \\
& = & (\lambda x^{\alpha^{\bullet}} y^{[\alpha]}.\mathrm{case}\ y^{[\alpha]}\{f \to [P, \upsilon[f/x^{\alpha}]]\}_{f\in\mathcal{S}_{\alpha}})\,[Q, \upsilon]\,[\![Q]\!]^{\upsilon} \\
& \to^{+}_{\beta\delta h} & [P, \upsilon[[\![Q]\!]^{\upsilon}/x^{\alpha}]]\,[[Q, \upsilon]/x^{\alpha}] \\
& = & \left[M', \upsilon\right]
\end{array}$$

In case $M = Y^{(\alpha\to\alpha)\to\alpha}P$, we have $M' = PM$ and:

$$\begin{array}{rcl}
[M, \upsilon] & = & Y^{(\alpha^{\bullet}\to\alpha^{\bullet})\to\alpha^{\bullet}}(\lambda x^{\alpha^{\bullet}}.\,[P, \upsilon]\,x^{\alpha^{\bullet}}[\![M]\!]^{\upsilon}) \\
& \to_{\beta\delta h} & (\lambda x^{\alpha^{\bullet}}.\,[P, \upsilon]\,x^{\alpha^{\bullet}}[\![M]\!]^{\upsilon})\,[M, \upsilon] \\
& \to_{\beta\delta h} & [P, \upsilon]\,[M, \upsilon]\,[\![M]\!]^{\upsilon} \\
& = & [PM, \upsilon] \\
& = & \left[M', \upsilon\right]
\end{array}$$

□

Corollary 39 Given a term M of type 0 and a valuation υ , $M \to^{*}_{\beta\delta h} aM_1M_2$ iff $[M, \upsilon] \to^{*}_{\beta\delta h} a^{[\![M]\!]^{\upsilon}}\,[M_1, \upsilon]\,[M_2, \upsilon]$.

Proof
The direction from left to right is a simple consequence of Lemma 38. For the direction from right to left, we use the well-known fact that a λ-term has a head normal form iff it can be head-reduced to a head normal form. Let us suppose that $[M, \upsilon]$ reduces to $a^{[\![M]\!]^{\upsilon}}P_1P_2$ in k steps of head-reduction. In case M has no head normal form, then, for $l = k + 1$ there is P such that $M \to^{l}_{\beta\delta h} P$ (where $\to^{l}_{\beta\delta h}$ denotes the relation of l steps of head reduction), but, then, by an iterative use of Lemma 38, we must have $[M, \upsilon] \to^{m}_{\beta\delta h} [P, \upsilon]$ with $k < l \leq m$ and we obtain a contradiction. In case M can be reduced in l reduction steps to bN_1N_2, then a simple use of Lemma 38 gives that $b = a$, $P_1 = [N_1, \upsilon]$ and $P_2 = [N_2, \upsilon]$. □

Theorem 40 *For every finitary model $\mathcal{S}$ and a closed term M of type 0: $BT([M,\emptyset]) = rBT_{\mathcal{S}}(M)$.*

Proof
This is a simple consequence of Corollary 39 □

Remark: If in the model $\mathcal{S}$ the divergence can be observed (as it is the case for GFP models and for the model $\mathcal{K}$, cf. Proposition 4) then in the translation above we could add the rule $[M,\upsilon] = \Omega$ whenever $[\![M]\!]^{\upsilon}$ denotes a diverging term. We would obtain a term which would always converge. A different construction for achieving the same goal is proposed in [Had12].

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 and thus avoids the unnecessary use of *case* construction. For this, the translation is now parametrized also with a stack of values from $\mathcal{S}$ so as 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 putting the λ-term in η-long form).

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

6 Conclusions

We have shown that a model-based approach to model-checking works for quite a big 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 [Kob09c] and step functions in models are in direct correspondence [SMGB12], 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 to mix a GFP model for Ω-blind automaton with the model $\mathcal{D}$ for detecting solvability.

The approach through 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 a Eilenberg like variety theory for lambda-terms [Eil74]. 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

[AC98] R. M. Amadio and P-L. Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[Aeh07] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[Bar84] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[BCOS10] C. Broadbent, A. Carayol, L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129, 2010.

[Bro12] C. H. Broadbent. The Limits of Decidability for First Order Logic on CPDA Graphs. In *STACS 2012*, volume 14 of *LIPIcs*, pages 589–600, 2012.

[Eil74] S. Eilenberg. *Automata, Languages and Machines.* Academic Press, New York, 1974.

[Had12] A. Haddad. IO vs OI in higher-order recursion schemes. In *FICS*, volume 77 of *EPTCS*, pages 23–30, 2012.

[HMOS08] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

[KO09] N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.

[Kob09a] N. Kobayashi. Higher-order program verification and language-based security. In *ASIAN*, volume 5913 of *LNCS*, pages 17–23. Springer, 2009.

[Kob09b] N. Kobayashi. Model-checking higher-order functions. In *PPDP*, pages 25–36. ACM, 2009.

[Kob09c] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

[Kob09d] N. Kobayashi. Types and recursion schemes for higher-order program verification. In *APLAS*, volume 5904 of *LNCS*, pages 2–3, 2009.

[Kob11] N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FOSSACS*, pages 260–274, 2011.

[LNOR10] M.M. Lester, R.P. Neatherway, C.-H. L. Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. Technical report, 2010.

[Ong06] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

[OT12] C.-H. Luke Ong and Takeshi Tsukada. Two-level game semantics, intersection types, and recursion schemes. In *ICALP (2)*, volume 7392 of *LNCS*, pages 325–336, 2012.

[SMGB12] S. Salvati, G. Manzonetto, M. Gehrke, and H. Barendregt. Loader and Urzyczyn are logically related. In *ICALP (2)*, LNCS, pages 364–376, 2012.

[SW11] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, volume 6756 of *LNCS*, pages 162–173, 2011.

[SW12] S. Salvati and I. Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In *RP*, volume 7550 of *LNCS*, pages 6–20, 2012.

[Wal12] Igor Walukiewicz. Simple models for recursive schemes. In *MFCS*, volume 7464 of *LNCS*, pages 49–60, 2012.