Typing Weak MSOL Properties

Sylvain Salvati and Igor Walukiewicz

CNRS, Université de Bordeaux, INRIA

Abstract. We consider λY-calculus as a non-interpreted functional programming language: the result of the execution of a program is its normal form that can be seen as the tree of calls to built-in operations. Weak monadic second-order logic (wMSO) is well suited to express properties of such trees. We give a type system for ensuring that the result of the execution of a λY-program satisfies a given wMSO property. In order to prove soundness and completeness of the system we construct a denotational semantics of λY-calculus that is capable of computing properties expressed in wMSO.

1 Introduction

Higher-order functional programs are more and more often used to write interactive applications. In this context it is important to reason about behavioral properties of programs. We present a kind of type and effect discipline [22] where a well-typed program will satisfy behavioral properties expressed in weak monadic second-order logic (wMSO).

We consider the class of programs written in the simply-typed calculus with recursion and finite base types: the λY-calculus. This calculus offers an abstraction of higher-order programs that faithfully represents higher-order control. The dynamics of an interaction of a program with its environment is represented by the Böhm tree of a λY-term that is a tree reflecting the control flow of the program. For example, the Böhm tree of the term $Yx.ax$ is the infinite sequence of a's, representing that the program does an infinite sequence of a actions without ever terminating. Another example is presented in Figure 1. A functional program for the factorial function is written as a λY-term Fct and the value of Fct applied to a constant c is calculated. Observe that all constants in Fct are non-interpreted. The Böhm tree semantics means call-by-name evaluation strategy. Nevertheless, call-by-value evaluation can be encoded, so can be finite data domains, and conditionals over them [18,13]. The approach is then to translate a functional program to a λY-term and to examine the Böhm tree it generates.

Since the dynamics of the program is represented by a potentially infinite tree, monadic second-order logic (MSOL) is a natural candidate for the language to formulate properties in. This logic is an extension of first-order logic with quantification over sets. MSOL captures precisely regular properties of trees [25], and it is decidable if the Böhm tree generated by a given λY-term satisfies a given property [23]. In this paper we will restrict to weak monadic second-order logic (wMSO). The difference is that in wMSO quantification is restricted to range

A. Pitts (Ed.): FOSSACS 2015, LNCS 9034, pp. 343–357, 2015.
DOI: 10.1007/978-3-662-46678-0_22

$$Factorial(x) \equiv \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x \cdot Factorial(x-1)$$
$$Fct \equiv Y F.\ \lambda x.\ \textbf{if } (z\ x)\ 1\ (m\ x\ (F(-\ x\ 1)))$$

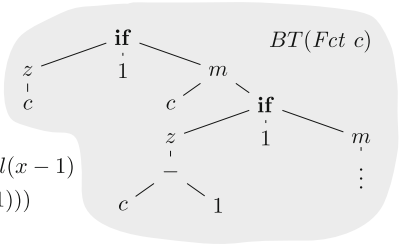

Fig. 1. Böhm tree of the factorial function

over finite sets. While wMSO is a proper fragment of MSO, it is sufficiently strong to express safety, reachability, and many liveness properties. Over sequences, that is degenerated trees where every node has one successor, wMSO is equivalent to full MSO.

The basic judgments we are interested in are of the form $BT(M) \vDash \alpha$ meaning that the result of the evaluation of M, i.e. the Böhm tree of M, has the property α formulated in wMSO. Going back to the example of the factorial function from Figure 1, we can consider a property: all computations that eventually take the "if" branch of the conditional are finite. This property holds in $BT(Fct\ c)$. Observe by the way that $BT(Fct\ c)$ is not regular – it has infinitely many non-isomorphic subtrees as the number of subtractions is growing. In general the interest of judgments of the form $BT(M) \vDash \alpha$ is to be able to express liveness and fairness properties of executions, like: "every *open* action is eventually followed by a *close* action", or that "there are infinitely many read actions". Various other verification problems for functional programs can be reduced to this problem [18,20,24,32,12].

Technically, the judgment $BT(M) \vDash \alpha$ is equivalent to determining whether a Böhm tree of a given λY-term is accepted by a given weak alternating automaton. This problem is known to be decidable thanks to the result of Ong [23], but here we present a denotational approach. Our two main contributions are:

- A construction of a finitary model for a given weak alternating automaton. The value of a term in this model determines if the Böhm tree of the term is accepted by the automaton. So verification is reduced to evaluation.
- Two type systems. A typing system deriving statements of the form "the value of a term M is bigger than an element d of the model"; and a typing system for dual properties. These typing systems use standard fixpoint rules and follow the methodology coined as *Domains in Logical Form* [1]. Thanks to the first item, these typing systems can directly talk about acceptance/rejection of the Böhm tree of a term by an automaton. These type systems are decidable, and every term has a "best" type that simply represents its value in the model.

Having a model and a type system has several advantages over having just a decision procedure. First, it makes verification compositional: the result for a term is calculated from the results for its subterms. In particular, it opens possibilities for a modular approach to the verification of large programs. Next, it enables semantic based program transformations as for example reflection of a given property in a given term [8,29,13]. It also implies the transfer theorem for wMSO [28] with a number of consequences offered by this theorem. Finally, models open a way to novel verification algorithms be it through evaluation, type system, or through hybrid algorithms using typing and evaluation at the same time [31]. We come back to these points in the conclusions.

Historically, Ong [23] has shown the decidability of the MSO theory of Böhm trees for all λY-terms. This result has been revisited in several different ways. Some approaches take a term of the base type, and unroll it to some infinite object: tree with pointers [23], computation of a higher-order pushdown automaton with collapse [14], a collection of typing judgments that are used to define a game [19], a computation of a Krivine machine [27]. Very recently Tsukada and Ong [33] have presented a compositional approach: a typing system is used to reduce the verification problem to a series of game solving problems. Another recent advance is given by Hofmann and Chen who provide a type system for verifying path properties of trees generated by first-order λY-terms [11]. In other words, this last result gives a typing system for verifying path properties of trees generated by deterministic pushdown automata. Compared to this last work, we consider the whole λY-calculus and an incomparable set of properties.

Already some time ago, Aehligh [2] has discovered an easy way to prove Ong's theorem restricted to properties expressed by tree automata with trivial acceptance conditions (TAC automata). The core of his approach can be formulated by saying that the verification problem for such properties can be reduced to evaluation in a specially constructed and simple model. Later, Kobayashi proposed a type system for such properties and constructed a tool based on it [18]. This in turn opened a way to an active ongoing research resulting in the steady improvement of the capacities of the verification tools [17,9,10,26]. TAC automata can express only safety properties. Our model and typing systems set the stage for practical verification of wMSO properties.

The model approach to verification of λY-calculus is quite recent. In [29] it is shown that simple models with greatest fixpoints capture exactly properties expressed with TAC automata. An extension is then proposed to allow one to detect divergence. The simplicity offered by models is exemplified by Haddad's recent work [13] giving simple semantic based transformations of λY-terms.

We would also like to mention two other quite different approaches to integrate properties of infinite behaviors into typing. Naik and Palsberg [21] make a connection between model-checking and typing. They consider only safety properties, and since their setting is much more general than ours, their type system is more complex too. Jeffrey [15,16] has shown how to incorporate Linear Temporal Logic into types using a much richer dependent types paradigm. The calculus

is intended to talk about control and data in functional reactive programming framework, and aims at using SMT solvers.

In the next section we introduce the main objects of our study: λY-calculus, and weak alternating automata. Section 3 presents the type system. Its soundness and completeness can be straightforwardly formulated for closed terms of atomic type. For the proof though we need a statement about all terms. This is where the model based approach helps. Section 4 describes how to construct models for wMSO properties. In Section 5 we come back to our type systems. The general soundness and completeness property we prove says that types can denote every element of the model, and the type systems can derive precisely the judgments that hold in the model (Theorem 3). In the conclusion section we mention other applications of our model. All proofs can be found in a long version of the paper [30].

2 Preliminaries

We quickly fix notations related to the simply typed λY-calculus and to Böhm trees. We then recall the definition of weak alternating automata on ranked trees. These will be used to specify properties of Böhm trees.

λY-calculus. The *set of types* $\mathcal{T}$ is constructed from a unique *basic type* o using a binary operation $\to$ that associates to the right. Thus o is a type and if A, B are types, so is $(A \to B)$. The order of a type is defined by: $order(o) = 0$, and $order(A \to B) = max(1 + order(A), order(B))$. We work with *tree signatures* that are finite sets of *typed constants of order at most* 1. Types of order 1 are of the form $o \to \cdots \to o \to o$ that we abbreviate $o^i \to o$ when they contain $i + 1$ occurrences of o. For convenience we assume that $o^0 \to o$ is just o. If Σ is a signature, we write $\Sigma^{(i)}$ for the set of constants of type $o^i \to o$. In examples we will often use constants of type $o \to o$ as this makes the examples more succinct. At certain times, we will restrict to the types o and $o^2 \to o$ that are representative for all the cases.

Simply typed λY-terms are built from the constants in the signature, and constants Y^A, Ω^A for every type A. These stand for the *fixpoint combinator* and *undefined term*, respectively. Apart from constants, for each type A there is a countable set of variables $x^A, y^A, \ldots$. Terms are built from these constants and variables using typed application and λ-abstraction. We shall write sequences of λ-abstractions $\lambda x_1 \ldots \lambda x_n.\ M$ with only one λ: either $\lambda x_1 \ldots x_n.\ M$, or even shorter $\lambda \boldsymbol{x}.\ M$. We will often write $Yx.M$ instead of $Y(\lambda x.M)$. Every λY-term can be written in this notation since YN has the same Böhm tree as $Y(\lambda x.Nx)$, and the latter term is $Yx.(Nx)$. We take for granted the operational semantics of the calculus given by β and δ reductions. The *Böhm tree* of a term M is obtained by reducing it until one reaches a term of the form $\lambda \boldsymbol{x}.N_0 N_1 \ldots N_k$ with N_0 a variable or a constant. Then $BT(M)$ is a tree having its root labeled by $\lambda \boldsymbol{x}.N_0$ and having $BT(N_1), \ldots, BT(N_k)$ as subtrees. Otherwise $BT(M) = \Omega^A$, where A is the type of M. Böhm trees are infinite normal forms of λY-terms. A Böhm

tree of a closed term of type o over a tree signature is a potentially infinite ranked tree: a node labeled by a constant a of type $o^i \to o$ has i successors (c.f. Figure 1).

Example. As an example take $(YF.\, N)a$ where $N = \lambda g.g(b(F(\lambda x.g(g\, x))))$. Both a and b have the type $o \to o$; while F has type $(o \to o) \to o$, and so does N. Observe that we are using a more convenient notation YF here. The Böhm tree of $(YF.N)a$ is $BT((YF.N)a) = aba^2ba^4b \ldots a^{2^n}b \ldots$ after every consecutive occurrence of b the number of occurrences of a doubles because of the double application of g inside N.

wMSO and Weak Alternating Automata. We will be interested in properties of trees expressed in weak monadic second-order logic. This is an extension of first-order logic with quantification over finite sets of elements. The interplay of negation and quantification allows the logic to express many infinitary properties. The logic is closed for example under constructs: "for infinitely many vertices a given property holds", "every path consisting of vertices having a given property is finite". From the automata point of view, the expressive power of the logic is captured by weak alternating automata. A *weak alternating automaton* accepts trees over a fixed tree signature Σ.

A *weak alternating tree automaton* over the signature Σ is:

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \{\delta_i\}_{i \in \mathcal{N}}, \rho : Q \to \mathcal{N} \rangle$$

where Q is a finite set of states, $q^0 \in Q$ is the initial state, ρ is the *rank function*, and $\delta_i : Q \times \Sigma^{(i)} \to \mathcal{P}(\mathcal{P}(Q)^i)$ is the transition function. For q in Q, we call $\rho(q)$ *its rank*. The automaton is *weak* in the sense that when $(S_1, \ldots, S_i)$ is in $\delta_i(q, a)$, then the rank of every q' in $\bigcup_{1 \le j \le i} S_j$ is not bigger than the rank of q, $\rho(q') \le \rho(q)$.

Observe that since Σ is finite, only finitely many δ_i are nontrivial. From the definition it follows that $\delta_2 : Q \times \Sigma^{(2)} \to \mathcal{P}(\mathcal{P}(Q) \times \mathcal{P}(Q))$ and $\delta_0 : Q \times \Sigma^{(0)} \to \{0, 1\}$. We will simply write δ without a subscript when this causes no ambiguity.

Automata will work on Σ-labeled binary trees that are partial functions $t : \mathcal{N}^* \stackrel{\cdot}{\to} \Sigma \cup \{\Omega\}$ such that the number successors of a node is determined by the label of the node. In particular, if $t(u) \in \Sigma^{(0)} \cup \{\Omega\}$ then u is a leaf.

The acceptance of a tree is defined in terms of *games* between two players that we call Eve and Adam. A *play* between Eve and Adam from some node v of a tree t and some state $q \in Q$ proceeds as follows. If v is a leaf and is labeled by some $c \in \Sigma^{(0)}$ then Eve wins iff $\delta_0(q, c)$ holds. If the node is labeled by Ω then Eve wins iff the rank of q is even. Otherwise, v is an internal node; Eve chooses a tuple of sets of states $(S_1, \ldots, S_i) \in \delta(q, t(v))$. Then Adam chooses S_j (for $j = 1, \ldots, i$) and a state $q' \in S_j$. The play continues from the j-th son of v and state q'. When a player is not able to play any move, he/she looses. If the play is infinite then the winner is decided by looking at ranks of states appearing on the play. Due to the weakness of $\mathcal{A}$ the rank of states in a play can never increase, so it eventually stabilizes at some value. Eve wins if this value is even. A tree t is *accepted* by $\mathcal{A}$ from a state $q \in Q$ if Eve has a winning strategy in the game started from the root of t and from q.

Automata with *trivial acceptance conditions*, as considered by Kobayashi [17], are obtained by requiring that all states have rank 0. Automata with co-trivial are just those whose all states have rank 1.

Observe that without a loss of generality we can assume that δ is monotone, i.e. if $(S_1, \ldots, S_i) \in \delta(q, a)$ then for every $(S'_1, \ldots, S'_i)$ such that $S_j \subseteq S'_j \subseteq \{q' : \rho(q') \leq \rho(q)\}$ we have $(S'_1, \ldots, S'_i) \in \delta(q, a)$. Indeed, adding the transitions needed to satisfy the monotonicity condition does not give Eve more winning possibilities.

An automaton defines a language of closed terms of type o whose Böhm trees it accepts from its initial state q^0:

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

3 Type Systems for wMSOL

In this section we describe the main result of the paper. We present a type system to reason about wMSO properties of Böhm trees of terms (a dual type system is presented in the appendix). We will rely on the equivalence of wMSO and weak alternating automata, and construct a type system for an automaton. For a fixed weak alternating automaton $\mathcal{A}$ we want to characterize the terms whose Böhm trees are accepted by $\mathcal{A}$, i.e. the set $L(\mathcal{A})$. The characterization will be purely type theoretic (cf. Theorem 1).

Fix an automaton $\mathcal{A} = \langle Q, \Sigma, q^0, \{\delta_i\}_{i \in \mathcal{N}}, \rho \rangle$. Let m be the maximal rank, i.e., the maximal value ρ takes on Q. For every $0 \leq k \leq m$ we write $Q_k = \{q \in Q : \rho(q) = k\}$ and $Q_{\leq k} = \{q \in Q : \rho(q) \leq k\}$.

The type system we propose is obtained by allowing the use of intersections inside simple types. This idea has been used by Kobayashi [18] to give a typing characterization for languages of automata with trivial acceptance conditions. We work with, more general, weak acceptance conditions, and this will be reflected in the stratification of types, and two fixpoint rules: greatest fixpoint rule for even strata, and the least fixpoint rule for odd strata.

First, we define the sets of intersection types. They are indexed by a rank of the automaton and by a simple type. Note that every intersection type will have a corresponding simple type; this is a crucial difference with intersection types characterizing strongly normalizing terms [4]. Letting $\mathrm{Types}^k_A = \bigcup_{0 \leq l \leq k} \mathrm{types}^l_A$ we define:

$$\mathrm{types}^k_o = \{q \in Q : \rho(q) = k\},\ \mathrm{types}^k_{A \to B} = \{T \to s : T \subseteq \mathrm{Types}^k_A \text{ and } s \in \mathrm{types}^k_B\}\ .$$

The difference with simple types is that now we have a set constructor that will be interpreted as the intersection of its elements.

When we write types_A or Types_A we mean types^m_A and Types^m_A respectively; where m is the maximal rank used by the automaton $\mathcal{A}$.

For $S \subseteq \mathrm{Types}^k_A$ and $T \subseteq \mathrm{types}^k_B$ we write $S \to T$ for $\{S \to t : t \in T\}$. Notice that $S \to T$ is included in $\mathrm{types}^k_{A \to B}$.

We now give subsumption rules that express the intuitive dependence between types. So as to make the connection with the model construction later, we have adopted an ordering of intersection types that is dual to the usual one.

$$\frac{S \subseteq T \subseteq Q}{S \sqsubseteq_0 T} \qquad \frac{\forall s \in S, \exists t \in T, s \sqsubseteq_A t}{S \sqsubseteq_A T} \qquad \frac{s = t}{s \sqsubseteq_0 t} \qquad \frac{T \sqsubseteq_A S \quad s \sqsubseteq_B t}{S \to s \sqsubseteq_{A\to B} T \to t}$$

Given $S \subseteq \text{Types}_{A\to B}$ and $T \subseteq \text{Types}_A$ we write $S(T)$ for the set $\{t : (U \to t) \in S \wedge U \sqsubseteq T\}$.

The typing system presented in Figure 2 derives judgments of the form $\Gamma \vdash M \geq S$ where Γ is an environment containing all the free variables of the term M, and $S \subseteq \text{Types}_A$ with A the type of M. As usual, an environment Γ is a finite list $x_1 \geq S_1, \ldots, x_n \geq S_n$ where $x_1, \ldots, x_n$ are pairwise distinct variables of type A_i, and $S_i \subseteq \text{Types}_{A_i}$. We will use a functional notation and write $\Gamma(x_i)$ for S_i. We shall also write $\Gamma, x \geq S$ with its usual meaning.

The rules in the first row of Figure 2 express standard intersection types dependencies: the axiom, the intersection rule and the subsumption rule. The rules in the second line are specific to our fixed automaton. The third line contains the usual rules for application and abstraction. The least fixpoint rule in the next line is standard. The greatest fixpoint rule in the last line is more intricate. It is allowed only on even strata. If taken for $k = 0$ the rule becomes the standard rule for the greatest fixpoint as the set T must be the empty set. For $k > 0$ the rule permits to incorporate T that is the result of the fixpoint computation on the lower stratum.

$$\frac{}{\Gamma, x \geq S \vdash x \geq S} \qquad \frac{\Gamma \vdash M \geq S \quad \Gamma \vdash M \geq T}{\Gamma \vdash M \geq S \cup T} \qquad \frac{\Gamma \vdash M \geq S \quad T \sqsubseteq S}{\Gamma \vdash M \geq T}$$

$$\frac{}{\Gamma \vdash c \geq \{q : \delta_o(q, c) \text{ holds}\}} \qquad \frac{(S_1, \ldots, S_i) \in \delta(a, q)}{\Gamma \vdash a \geq \{S_1 \to \cdots \to S_i \to q\}}$$

$$\frac{\Gamma \vdash M \geq S \quad \Gamma \vdash N \geq T}{\Gamma \vdash MN \geq S(T)} \qquad \frac{S \subseteq \text{Types}^k, T \subseteq \text{types}^k \quad \Gamma, x \geq S \vdash M \geq T}{\Gamma \vdash \lambda x.M \geq S \to T}$$

$$\frac{\Gamma \vdash (\lambda x.M) \geq S \quad \Gamma \vdash (Yx.M) \geq T}{\Gamma \vdash Yx.M \geq S(T)}\ Y \textit{ odd}$$

$$\frac{S \subseteq \text{types}_A^{2k}, \quad T \subseteq \text{Types}_A^{2k-1}, \quad \Gamma \vdash \lambda x.M \geq (S \cup T) \to S \quad \Gamma \vdash Yx.M \geq T}{\Gamma \vdash Yx.M \geq S \cup T}\ Y \textit{ even}$$

Fig. 2. Type system

The main result of the paper says that the typing in this system is equivalent to accepting with our fixed weak alternating automaton.

Theorem 1. *For every closed term M of type o and every state q of $\mathcal{A}$: the judgment $\vdash M \geq q$ is derivable iff $\mathcal{A}$ accepts $BT(M)$ from q.*

Since there are finitely many types, this typing system is decidable. As we will see in the following example, this type system allows us to prove in a rather simple manner properties of Böhm trees that are beyond the reach of trivial automata. Compared to Kobayashi and Ong type system [19], the fixpoint typing rules we propose avoid the use of an external solver for a parity game. Our type system makes it also evident what is the meaning of higher-order terms with free variables. In the example below we use fixpoint rules on terms of order 2.

Example 2. Consider the term $M = (YF.N)a$ where $N = \lambda g.g(b(F(\lambda x.g(g\,x))))$. As we have seen on page 347, $BT(M) = aba^2ba^4b\ldots a^{2^n}b\ldots$. We show with typing that there are infinitely many occurrences of b in $BT(M)$. To this end we take an automaton has states $Q = \{q_1, q_2\}$, and works over the signature that contains a and b. The transitions of the automaton are:

$$\delta(q_1, a) = \{q_1\} \quad \delta(q_2, a) = \{q_1, q_2\} \quad \delta(q_1, b) = \emptyset \quad \delta(q_2, b) = q_2$$

The ranks of states are indicated by their subscripts. Starting with state q_2, the automaton only accepts sequences that contain infinitely many b's. So our goal is to derive $\vdash (YF.N)a \geq q_2$. First observe that from the definition of the transitions of the automaton we get axioms:

$$\frac{}{\vdash a \geq q_1 \to q_1} \qquad \frac{}{\vdash a \geq \{q_1, q_2\} \to q_2} \qquad \frac{}{\vdash b \geq \emptyset \to q_1} \qquad \frac{}{\vdash b \geq q_2 \to q_2}$$

Looking at the typings of a, we can see that we will get our desired judgment from the application rule if we prove:

$$\vdash YF.N \geq S \qquad \text{where } S \text{ is } \{q_1 \to q_1, \{q_1, q_2\} \to q_2\} \to q_2.$$

To this end, we apply subsumption rule and the greatest fixpoint rule:

$$\dfrac{\dfrac{\vdash \lambda F.N \geq (S \cup T) \to S \quad \vdash YF.N \geq T}{\vdash YF.N \geq S \cup T}\, Y\ even}{\vdash YF.N \geq S} \qquad \text{where } T = \{(q_1 \to q_1) \to q_1\}$$

The derivation of the top right judgment uses the least fixpoint rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{g \geq q_1 \to q_1 \vdash g \geq q_1 \to q_1} \qquad g \geq q_1 \to q_1 \vdash b(F(\lambda x.g(g\,x))) \geq q_1}{g \geq q_1 \to q_1 \vdash g(b(F(\lambda x.g(g\,x)))) \geq q_1}}{\vdash \lambda F \lambda g.g(b(F(\lambda x.g(g\,x)))) \geq \emptyset \to (q_1 \to q_1) \to q_1}}{\vdash YF.N \geq (q_1 \to q_1) \to q_1}\, Y\ odd$$

We have displayed only one of the two premises of the $Y odd$ rule since the other is of the form $\geq \emptyset$ so it is vacuously true. The top right judgment is derivable directly from the axiom on b. The derivation of the remaining judgment $\vdash \lambda F.N \geq (S \cup T) \to S$ is as follows.

$$\dfrac{\dfrac{}{\Gamma \vdash g \geq \{q_1, q_2\} \to q_2} \quad \Gamma \vdash b(F(\lambda x.g(g\,x))) \geq q_1, q_2}{\dfrac{\Gamma \vdash g(b(F(\lambda x.g(g\,x)))) \geq q_2}{\vdash \lambda F \lambda g.g(b(F(\lambda x.g(g\,x)))) \geq (S \cup T) \to S}}$$

where Γ is $F \geq S \cup T, g \geq \{q_1 \to q_1, \{q_1, q_2\} \to q_2\}$. So the upper left judgment is an axiom. The other judgment on the top is an abbreviation of two judgments: one to show $\geq q_1$ and the other one to show $\geq q_2$. These two judgments are proven directly using application and intersection rules.

4 Models for Weak Automata

This section presents the model that captures wMSO properties. We assume basic knowledge about domain theory. More specifically, we shall work with (finite) complete lattices and with monotone functions between complete lattices. Given two complete lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ we write $\mathrm{mon}[\mathcal{L}_1 \mapsto \mathcal{L}_2]$ for the complete lattice of monotone functions between $\mathcal{L}_1$ and $\mathcal{L}_2$. We construct a model that captures the language defined by a weak automaton: this model depends only on the states of the automaton and their ranks. The transitions of the automaton will be encoded in the interpretation of constants.

The challenge in this construction comes from the fact that simply using the least or greatest fixpoints is not sufficient. Indeed, we have shown in [29] that extremal fixpoints in finitary models of λY-calculus capture precisely boolean combinations of properties expressed by automata with trivial acceptance conditions. The structure of a weak automaton will help us here. For the sake of the discussion let us fix an automaton $\mathcal{A}$, and let $\mathcal{A}_{\leq k}$ stand for $\mathcal{A}$ restricted to states of rank at most k. Ranks stratify the automaton: transitions for states of rank k depend only on states of rank at most k. We will find this stratification in our model too. The interpretation of a term at stratum k will give us the complete information about the behaviour of the term with respect to $\mathcal{A}_{\leq k}$. Stratum $k+1$ will refine this information. Since in a run the ranks cannot increase, the information calculated at stratum $k+1$ does not change what we already know about $\mathcal{A}_{\leq k}$. Abstract interpretation tells us that refinements of models are obtained via Galois connections which are instrumental in our construction. In our model, every element in the stratum k is refined into a complete lattice in the stratum $k+1$ (cf. Figure 3). Therefore we will be able to define the interpretations of fixpoints by taking at stratum k the least or the greatest fixpoint depending on the parity of k. In the whole model, the fixpoint computation will perform a sort of zig-zag as represented in Figure 4.

We fix a finite set of states Q and a ranking function $\rho : Q \to \mathcal{N}$. Let m be the maximal rank, i.e., the maximal value ρ takes on Q. Recall that for every $0 \leq k \leq m$ we let $Q_k = \{q \in Q : \rho(q) = k\}$ and $Q_{\leq k} = \{q \in Q : \rho(q) \leq k\}$.

We define by induction on $k \leq m$ an applicative structure $\mathcal{D}^k = (\mathcal{D}^k_A)_{A \in \mathrm{types}}$ and a logical relation $\mathcal{L}^k$ (for $0 < k$) between $\mathcal{D}^{k-1}$ and $\mathcal{D}^k$. For $k = 0$, the model $\mathcal{D}^0$ is just the model of monotone functions over the powerset of Q_0 with

$\mathcal{D}^0_o = \mathcal{P}(Q_0)$ and $\mathcal{D}^0_{A\to B} = \mathrm{mon}[\mathcal{D}^0_A \mapsto \mathcal{D}^0_B]$. For $k > 0$, we define $\mathcal{D}^k$ by means of $\mathcal{D}^{k-1}$ and a logical relation $\mathcal{L}^k$:

$$\begin{aligned}
\mathcal{D}^k_o =& \mathcal{P}(Q_{\leq k}) \quad \mathcal{L}^k_o = \{(R,P) \in \mathcal{D}^{k-1}_o \times \mathcal{D}^k_o \ : \ R = P \cap Q_{\leq(k-1)}\},\\
\mathcal{L}^k_{A\to B} =& \{(f_1,f_2) \in \mathcal{D}^{k-1}_{A\to B} \times \mathrm{mon}[\mathcal{D}^k_A \mapsto \mathcal{D}^k_B] :\\
& \qquad\qquad \forall (g_1,g_2) \in \mathcal{L}^k_A.\ (f_1(g_1), f_2(g_2)) \in \mathcal{L}^k_B\}\\
\mathcal{D}^k_{A\to B} =& \{f_2 \ : \ \exists f_1 \in \mathcal{D}^{k-1}_{A\to B}.\ (f_1,f_2) \in \mathcal{L}^k_{A\to B}\}
\end{aligned}$$

Observe that $\mathcal{D}^k_A$ is defined by a double induction: the outermost on k and the auxiliary induction on the size of the type. Since $\mathcal{L}^k$ is a logical relation between $\mathcal{D}^{k-1}$ and $\mathcal{D}^k$, each $\mathcal{D}^k$ is an applicative structure. As $\mathcal{D}^{k-1}_o = \mathcal{P}(Q_{\leq(k-1)})$, the refinements of elements R in $\mathcal{D}^{k-1}_o$ are simply the sets P in $\mathcal{P}(Q_{\leq k})$ so that $R = P \cap Q_{\leq(k-1)}$. This explains the definition of $\mathcal{L}^k_o$. For higher types, $\mathcal{L}^k$ is defined as it is usual for logical relations. Notice that $\mathcal{D}^k_{A\to B}$ is the subset of the monotone functions e from $\mathcal{D}^k_A$ to $\mathcal{D}^k_B$ for which there exist an element d in $\mathcal{D}^{k-1}_{A\to B}$ so that (d,e) is in $\mathcal{L}^k_{A\to B}$; that is we only keep those monotone functions that correspond to refinements of elements in $\mathcal{D}^{k-1}_{A\to B}$.

Remarkably this construction puts a lot of structure on $\mathcal{D}^k_A$. The first thing to notice is that for each type A, $\mathcal{D}^k_A$ is a complete lattice. Given d in $\mathcal{D}^{k-1}_A$, we write $\mathcal{L}^k_A(d)$ for the set $\{e \in \mathcal{D}^k_A : (d,e) \in \mathcal{L}^k_A\}$. For each d, we have that $\mathcal{L}^k_A(d)$ is a complete lattice and that moreover, for d_1 and d_2 in $\mathcal{D}^{k-1}_A$, $\mathcal{L}^k_A(d_1)$ and $\mathcal{L}^k_A(d_2)$ are isomorphic complete lattices. We write $d^{\uparrow\vee}$ and $d^{\uparrow\wedge}$ respectively for the greatest and the least elements of $\mathcal{L}^k_A(d)$. Finally, for each element e in $\mathcal{D}^k_A$, there is a unique d so that (d,e) is in $\mathcal{L}^k_A$, we write $e^{\downarrow}$ for that element. Figure 3 represents schematically the essential properties of $\mathcal{D}^k_A$.

The formalization of the intuition that $\mathcal{D}^k_A$ is a refinement of $\mathcal{D}^{k-1}_A$ is given by the fact that the mappings $(\cdot)^{\downarrow}$ and $(\cdot)^{\uparrow\vee}$ form a Galois connection between $\mathcal{D}^k_A$ and $\mathcal{D}^{k-1}_A$ and that $(\cdot)^{\downarrow}$ and $(\cdot)^{\uparrow\wedge}$ form a Galois connection between $\mathcal{D}^{k-1}_A$ and $\mathcal{D}^k_A$.

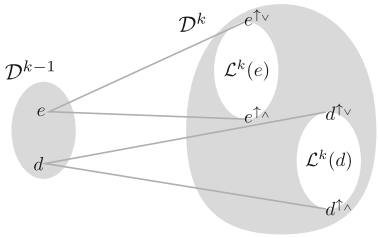

Fig. 3. Relation between models $\mathcal{D}^{k-1}$ and $\mathcal{D}^k$. Every element in $\mathcal{D}^{k-1}$ is related to a sub-lattice of elements in $\mathcal{D}^k$.

We can now define fixpoint operators in every applicative structure $\mathcal{D}^k$.

Definition 1. *For* $f \in \mathcal{D}^0_{A\to A}$ *we define* $\mathrm{fix}^0_A(f) = \bigwedge\{f^n(\top^0) : n \geq 0\}$. *For* $0 < 2k \leq m$ *and* $f \in \mathcal{D}^{2k}_{A\to A}$ *we define*

$$\mathrm{fix}^{2k}_A(f) = \bigwedge\{f^n(e) : n \geq 0\} \text{ where } e = (\mathrm{fix}^{2k-1}_A(f^{\downarrow}))^{\uparrow_\vee}$$

For $0 < 2k+1 \leq m$ *and* $f \in \mathcal{D}^{2k+1}_{A\to A}$ *we define*

$$\mathrm{fix}^{2k+1}_A(f) = \bigvee\{f^n(d) : n \geq 0\} \text{ where } d = (\mathrm{fix}^{2k}_A(f^{\downarrow}))^{\uparrow_\wedge}$$

Observe that, for even k, e is obtained with $(\cdot)^{\uparrow_\vee}$; while for odd k, $(\cdot)^{\uparrow_\wedge}$ is used.

The intuitive idea behind the definition of the fixpoint is presented in Figure 4. On stratum 0 it is just the greatest fixpoint. Then this greatest fixpoint is lifted to stratum 1, and the least fixpoint computation is started from it. The result is then lifted to stratum 2, and once again the greatest fixpoint computation is started, and so on. The Galois connections between strata guarantee that this process makes sense.

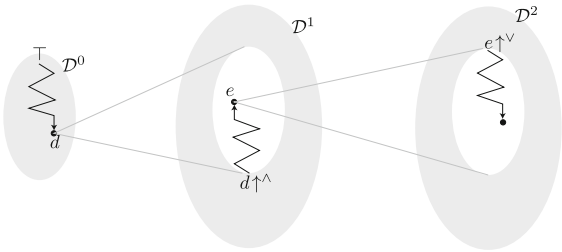

Fig. 4. A computation of a fixpoint: it starts in $\mathcal{D}^0$, and then the least and the greatest fixpoints alternate

Equipped with the interpretation of fixpoints given by Definition 1 the applicative structure $\mathcal{D}^k$ is a model of the λY-calculus. In particular, two terms that are $\beta\delta$-convertible have the same interpretation in that model. A constant c in $\Sigma^{(i)}$, is then interpreted as the function $f_{k,c}$ of $\mathcal{D}^k_{o^i\to o}$ so that for every $S_1, \ldots, S_i \subseteq \mathcal{D}^k_o$, $f_{k,c}(S_1, \ldots, S_i) = \{q \in \mathcal{D}^k_o : (S_1, \ldots, S_i) \in \delta(c, q)\}$. Observe that the identity $(f_{k+1,c})^{\downarrow} = f_{k,c}$ holds. Moreover if we let $\mathcal{A}(M) = \{q \in Q : \mathcal{A} \text{ accepts } BT(M) \text{ from } q\}$ be the set of states from which $\mathcal{A}$ accepts the tree $BT(M)$, then we have that:

Theorem 2. *For every closed term* M *of type* 0*, and for every* $0 \leq k \leq m$ *we have:* $[\![M]\!]^k = \mathcal{A}(M) \cap Q_{\leq k}$.

The two directions of Theorem 2 are proved using different techniques. The left to right inclusion uses a rather simple unrolling. The other inclusion is proved using a standard technique based on logical relations.

5 From Models to Type Systems

We are now in a position to show that our type system from Figure 2 can reason about the values of λY-terms in a stratified model (Theorem 3). Thanks to Theorem 2 this means that the type system can talk about the acceptance of the Böhm tree of a term by the automaton. This implies soundness and completeness of our type system, Theorem 1.

Throughout this section we work with a fixed signature Σ and a fixed weak alternating automaton $\mathcal{A} = \langle Q, \Sigma, q^0, \delta_o, \delta_{o^2 \to o}, \rho \rangle$. As in the previous section, for simplicity of notations we will assume that the constants in the signature are of type o or $o \to o \to o$. We will also prefer the notation $Yx.M$ to $Y(\lambda x.M)$.

The arrow constructor in types will be interpreted as a step function in the model. Step functions are particular monotone functions from a lattice $\mathcal{L}_1$ to a lattice $\mathcal{L}_2$. For d in $\mathcal{L}_1$ and e in $\mathcal{L}_2$, the *step function* $d \rightharpoonup e$ is defined by:

$$(d \rightharpoonup e)(h) = e \text{ when } d \leq h \quad \bot \text{ otherwise}$$

Types can be meaningfully interpreted at every level of the model. So $[\![t]\!]^l$ will denote the interpretation of t in $\mathcal{D}^l$ defined as follows.

$$[\![q]\!]^l = \{q\} \text{ if } \rho(q) \leq l, \emptyset \text{ otherwise}$$
$$[\![S]\!]^l = \bigvee \{ [\![t]\!]^l : t \in S \} \qquad \text{for } S \subseteq \text{Types}_A$$
$$[\![T \to s]\!]^l = [\![T]\!]^l \rightharpoonup [\![s]\!]^l \qquad \text{for } (T \to s) \in \text{Types}_A$$

Actually every element of $\mathcal{D}^l$ is the image of some type via $[\![\cdot]\!]^l$: types are syntactic representations of the model. The next theorem is the main technical result of the paper. It says that the type system can derive all lower-approximations of the meanings of terms in the model. For an environment Γ, we write $[\![\Gamma]\!]^k$ for the valuation such that $[\![\Gamma]\!]^k(x) = [\![\Gamma(x)]\!]^k$.

Theorem 3. *For $k = 0, \ldots, m$ and $S \subseteq \text{Types}^k$: $[\![M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k$ iff $\Gamma \vdash M \geq S$ is derivable.*

The above theorem implies Theorem 1 stating soundness and completeness of the type system. Indeed, let us take a closed term M of type o, and a state q of our fixed automaton $\mathcal{A}$. Theorem 2 tells us that $[\![M]\!] = \mathcal{A}(M)$; where $\mathcal{A}(M)$ is the set of states from which $\mathcal{A}$ accepts $BT(M)$. So $\vdash M \geq q$ is derivable iff $[\![M]\!] \supseteq \{q\}$ iff $q \in \mathcal{A}(M)$.

One may ask if it is also possible to reason about over-approximations of the value of a term, i.e. about statements of the form $[\![M]\!]^k_{[\![\Gamma]\!]^k} \leq d$. This is indeed possible thanks to the dualities of the model. It is enough to dualize the type system: restricting the rule for greatest fixpoint on odd ranks instead of even ones, taking the dual subsumption order for the types, and typing constant with the transitions of the dual weak alternating automaton. This dual system is presented in the appendix. It derives judgments of the form: $\Gamma \vdash M \not\geq S$ since the interpretation of S is also dualized. Without going into details of this dualization we can state the following theorem.

Theorem 4. *For every closed term M of type o and every state q of $\mathcal{A}$: the judgment $\vdash M \not\geq q$ is derivable iff $\mathcal{A}$ does not accept $BT(M)$ from q.*

Together the type system and its dual give a precise characterization of $[\![M]\!] = L(\mathcal{A})$ that is the set of states from which our fixed automaton $\mathcal{A}$ accepts $BT(M)$.

Corollary 1. *For a closed term M of type o, $[\![M]\!] = [\![S]\!]$ iff both $\vdash M \geq S$ and $\vdash M \not\geq (Q - S)$.*

6 Conclusions

We have shown how to construct a model for a given weak alternating tree automaton so that the value of a term in the model determines if the Böhm tree of the term is accepted by the automaton. Our construction builds on ideas from [29] but requires to bring out the modular structure of the model. This structure is very rich, as testified by Galois connections. This structure allows us to derive type systems for wMSO properties following the "domains in logical form" approach.

The type systems are relatively streamlined: the novelty is the stratification of types used to restrict applicability of the greatest fixpoint rule. In comparison, Kobayashi and Ong [19] use a type system only as an intermediate formalism to obtain a game whose solution answers the model-checking problem. Their type system handles only closed terms of type o. It does not have a rule for lambda-abstraction, nor for fixpoints (that are handled via games). Tsukada and Ong have recently proposed a higher-order analogue of this system [33]. Even in this new approach the fixpoint is still handled by games, and the algorithmic properties of the model behind their system are not investigated. While our approach applies only to wMSO, our model is simply based on functions over finite sets with standard application operation.

Typing in our system is decidable, actually the height of the derivation is bounded by the size of the term. Yet the width can be large, that is unavoidable given that the typability is n-Exptime hard for terms of order n [31]. Due to the correspondence of the typing with semantics, every term has a "best" type.

While the paper focuses on typing, our model construction can be also used in other contexts. It allows us to immediately deduce reflection [8] and transfer [28] theorems for wMSO. Our techniques used to construct models and prove their correctness rely on usual techniques of domain theory [3], offering an alternative, and arguably simpler, point of view to techniques based on unrolling.

The idea behind the reflection construction is to transform a given term so that at every moment of its evaluation every subterm "knows" its meaning in the model. In [8] this property is formulated slightly differently and is proved using a detour to higher-order pushdown automata. Recently Haddad [13] has given a direct proof for all MSO properties. The proof is based on some notion of applicative structure that is less constrained than a model of the λY-calculus. One could apply his construction, or take the one from [29].

The transfer theorem says that for a fixed finite vocabulary of terms, an MSOL formula φ can be effectively transformed into an MSOL formula $\widehat{\varphi}$ such that for

every term M of type 0 over the fixed vocabulary: M satisfies $\widehat{\varphi}$ iff the Böhm tree of M satisfies φ. Since the MSO theory of a term, that is a finite graph, is decidable, the transfer theorem implies decidability of MSO theory of Böhm trees of λY-terms. As shown in [28] it gives also a number of other results.

A transfer theorem for wMSO can be deduced from our model construction. For every wMSO formula φ we need to find a formula $\widehat{\varphi}$ as above. For this we transform φ into a weak alternating automaton $\mathcal{A}$, and construct a model $\mathcal{D}_\varphi$ based on $\mathcal{A}$. Thanks to the restriction on the vocabulary, it is quite easy to write for every element d of the model $\mathcal{D}_\varphi$ a wMSO formula α_d such that for every term M of type 0 in the restricted vocabulary: $M \vDash \alpha_d$ iff $[\![M]\!]^{\mathcal{D}_\varphi} = d$. The formula $\widehat{\varphi}$ is then just a disjunction $\bigvee_{d\in F} \alpha_d$, where F is the set elements of $\mathcal{D}_\varphi$ characterizing terms whose Böhm tree satisfies φ.

The fixpoints in our models are non-extremal: they are neither the least nor the greatest fixpoints. From [29] we know that this is unavoidable. We are aware of very few works considering such cases. Our models are an instance of cartesian closed categories with internal fixpoint operation as studied by Bloom and Esik [6]. Our model satisfies not only Conway identities but also a generalization of the *commutative axioms* of iteration theories [5]. Thus it is possible to give semantics to the infinitary λ-calculus in our models. It is an essential step towards obtaining an algebraic framework for weak regular languages [7].

References

1. Abramsky, S.: Domain theory in logical form. Ann. Pure Appl. Logic 51(1-2), 1–77 (1991)
2. Aehlig, K.: A finite semantics of simply-typed lambda terms for infinite runs of automata. Logical Methods in Computer Science 3(1), 1–23 (2007)
3. Amadio, R.M., Curien, P.-L.: *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science, vol. 46. Cambridge University Press (1998)
4. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. J. Symb. Log. 4, 931–940 (1983)
5. Bloom, S.L., Ésik, Z.: Iteration Theories: The Equational Logic of Iterative Processes. EATCS Monographs in Theoretical Computer Science. Springer (1993)
6. Bloom, S.L., Ésik, Z.: Fixed-point operations on CCC's. part I. Theoretical Computer Science 155, 1–38 (1996)
7. Blumensath, A.: An algebraic proof of Rabin's tree theorem. Theor. Comput. Sci. 478, 1–21 (2013)
8. Broadbent, C., Carayol, A., Ong, L., Serre, O.: Recursion schemes and logical reflection. In: LICS, pp. 120–129 (2010)
9. Broadbent, C.H., Carayol, A., Hague, M., Serre, O.: C-shore: a collapsible approach to higher-order verification. In: ICFP, pp. 13–24. ACM (2013)
10. Broadbent, C.H., Kobayashi, N.: Saturation-based model checking of higher-order recursion schemes. In: CSL. LIPIcs, vol. 23, pp. 129–148. Schloss Dagstuhl (2013)
11. Chen, W., Hofmann, M.: Buchi abstraction. In: LICS (2014) (to appear)
12. Grabowski, R., Hofmann, M., Li, K.: Type-based enforcement of secure programming guidelines — code injection prevention at SAP. In: Barthe, G., Datta, A., Etalle, S. (eds.) FAST 2011. LNCS, vol. 7140, pp. 182–197. Springer, Heidelberg (2012)

13. Haddad, A.: Model checking and functional program transformations. In: FSTTCS. LIPIcs, vol. 24, pp. 115–126 (2013)
14. Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: LICS, pp. 452–461. IEEE Computer Society (2008)
15. Jeffrey, A.S.A.: LTL types FRP: Linear-time Temporal Logic propositions as types, proofs as functional reactive programs. In: ACM Workshop Programming Languages meets Program Verification (2012)
16. Jeffrey, A.S.A.: Functional reactive types. In: LICS (2014) (to appear)
17. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: POPL, pp. 416–428 (2009)
18. Kobayashi, N.: Model checking higher-order programs. J. ACM 60(3), 20–89 (2013)
19. Kobayashi, N., Ong, L.: A type system equivalent to modal mu-calculus model checking of recursion schemes. In: LICS, pp. 179–188 (2009)
20. Kobayashi, N., Tabuchi, N., Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification. In: POPL, pp. 495–508 (2010)
21. Naik, M., Palsberg, J.: A type system equivalent to a model checker. ACM Trans. Program. Lang. Syst. 30(5) (2008)
22. Nielson, F., Riis Nielson, H.: Type and effect systems. In: Olderog, E.-R., Steffen, B. (eds.) Correct System Design. LNCS, vol. 1710, pp. 114–136. Springer, Heidelberg (1999)
23. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS, pp. 81–90 (2006)
24. Ong, C.-H.L., Ramsay, S.: Verifying higher-order programs with pattern-matching algebraic data types. In: POPL, pp. 587–598 (2011)
25. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the AMS 141, 1–23 (1969)
26. Ramsay, S.J., Neatherway, R.P., Ong, C.-H.L.: A type-directed abstraction refinement approach to higher-order model checking. In: POPL, pp. 61–72. ACM (2014)
27. Salvati, S., Walukiewicz, I.: Krivine machines and higher-order schemes. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 162–173. Springer, Heidelberg (2011)
28. Salvati, S., Walukiewicz, I.: Evaluation is MSOL-compatible. In: FSTTCS. LIPIcs, vol. 24, pp. 103–114 (2013)
29. Salvati, S., Walukiewicz, I.: Using models to model-check recursive schemes. In: Hasegawa, M. (ed.) TLCA 2013. LNCS, vol. 7941, pp. 189–204. Springer, Heidelberg (2013)
30. alvati, S., Walukiewicz, I.: Typing weak MSOL properties (2014), `https://hal.archives-ouvertes.fr/hal-01061202`
31. Terui, K.: Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In: RTA. LIPIcs, vol. 15, pp. 323–338. Schloss Dagstuhl (2012)
32. Tobita, Y., Tsukada, T., Kobayashi, N.: Exact flow analysis by higher-order model checking. In: Schrijvers, T., Thiemann, P. (eds.) FLOPS 2012. LNCS, vol. 7294, pp. 275–289. Springer, Heidelberg (2012)
33. Tsukada, T., Ong, C.-H.L.: Compositional higher-order model checking via ω-regular games over Böhm trees. In: LICS (to appear, 2014)