Logical Methods in Computer Science
Vol. 13(1:14)2017, pp. 1–27
www.lmcs-online.org

Submitted Nov. 16, 2015
Published Mar. 23, 2017

TYPING WEAK MSOL PROPERTIES

SYLVAIN SALVATI [a] AND IGOR WALUKIEWICZ [b]

[a] INRIA, LaBRI, 351, cours de la Libération F-33405 Talence France
e-mail address: sylvain.salvati@labri.fr

[b] CNRS, LaBRI, 351, cours de la Libération F-33405 Talence France
e-mail address: igw@labri.fr

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 (wMSOL) 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 wMSOL 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 wMSOL.

1. Introduction

Higher-order functional programs are more and more frequently 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 [26] where a well-typed program will satisfy behavioral properties expressed in weak monadic second-order logic (wMSOL).

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 reflects the call-by-name evaluation strategy. Nevertheless, the call-by-value evaluation can be encoded, so can be finite data domains and conditionals over them [15, 20, 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 [29], and it is decidable if the Böhm tree generated by a

Key words and phrases: higher-order model checking, weak monadic second order logic, simply typed lambda-Y-calculus, denotational semantics, recognizability, finite state methods.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-13(1:14)2017

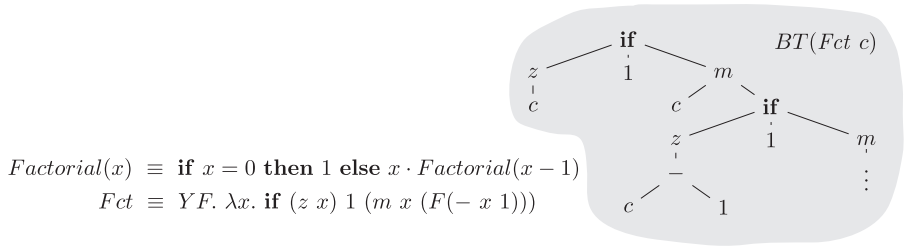

Figure 1: Böhm tree of the factorial function

given λY-term satisfies a given property [27]. In this paper we will restrict to weak monadic second-order logic (wMSOL). The difference is that in wMSOL quantification is restricted to range over finite sets. While wMSOL is a proper fragment of MSOL, it is sufficiently strong to express safety, reachability, and many liveness properties. Over sequences, that is, degenerated trees where every node has one successor, wMSOL is equivalent to full MSOL.

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 wMSOL. Going back to the example of the factorial function from Figure 1, we can consider a property: all computations that eventually take the middle branch in a node labeled by "if" 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 occurring in the left branches of "if" nodes is growing with the depth of those nodes. In general, the interest of judgments of the form $BT(M) \vDash \alpha$ is due to their ability 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 [20, 22, 28, 37, 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 [27], but we hope that the denotational approach we are pursuing here brings additional benefits. Our two main contributions are:

- A construction of a finitary model for a given weak alternating automaton. The ranking condition on the automaton is lifted to the denotational model and reflected in the alternation of the least and greatest fixpoints. 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 in the model.
- 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, 34, 13]. It also implies the transfer theorem for wMSOL [33] 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 [36]. We come back to these points in the conclusions.

Related work. Historically, Ong [27] has shown the decidability of the MSOL 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 [27], computation of a higher-order pushdown automaton with collapse [14], a collection of typing judgments that are used to define a game [21], a computation of a Krivine machine [32]. Recently, Tsukada and Ong [38] have presented a compositional approach: they give a typing system where the notion of a derivation is standard, their types are extended with annotations, and the fixpoint combinator is defined via game on these types with annotations. We will comment more on the relation with this work after we introduce our type system, as well as in the conclusions. Another recent advance is given by Hofmann and Chen [11] who provide a type system for verifying path properties of trees generated by first-order λY-terms. 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, Aehlig [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 [20]. This in turn opened a way to an active ongoing research resulting in the steady improvement of the capacities of the verification tools [19, 9, 10, 30]. TAC automata can express only safety properties. Our models consist of layers of models used by Aehlig, and our type system is a layered version of Kobayashi's system. This close relation to the model and type system for trivial properties, makes us hope that our model and typing system can be useful for practical verification of wMSOL properties.

The model approach to verification of λY-calculus is quite recent. In [34] 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 [25] 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 [17, 18] 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.

Organization of the paper. 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 wMSOL 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 5.5). In the conclusion section we mention other applications of our model.

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. Finally, we introduce the notion of the greatest fixpoint models for the λY-calculus. This notion allows us to adapt the definition of recognizability from language theory, so models can be used to define sets of terms. These sets of terms are closed under the reduction rules of the λY-calculus. We recall the characterization, in terms of automata, of the sets of terms recognizable by the greatest fixpoint models.

The *set $\mathcal{T}$ of types of λY-calculus* is constructed from a unique *basic type* o using a binary operation $\to$ that associates to the right[1]. Thus o is a type and if A, B are types, so is $(A \to B)$. The order of a type is defined by: $\mathit{order}(o) = 0$, and $\mathit{order}(A \to B) = \mathit{max}(1 + \mathit{order}(A), \mathit{order}(B))$. We work with *tree signatures* that are finite sets of *typed constants of order at most* 1. Types of order 1 are of the form $o \to \cdots \to o \to o$ that we abbreviate $o^i \to o$ when they contain $i+1$ occurrences of o. For convenience we assume that $o^0 \to o$ is just o. If Σ is a signature, we write $\Sigma^{(i)}$ for the set of constants of type $o^i \to o$. 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. The fixpoint combinators allows to have have computations with a recursion. The undefined terms represent diverging computation, but also, at a technical level are used to construct finite approximations of infinite computations. 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: if M has type $A \to B$ and N has type A, then (MN) has type B; and λ-abstraction: if M has type B then $(\lambda x^A.M)$ has type $A \to B$. We shall remove unnecessary parentheses, in particular, we write sequences of applications $((N_0 N_1) \ldots N_n)$ as $N_0 \ldots N_n$ and we write sequences of λ-abstractions $\lambda x_1 \ldots \lambda x_n.\ M$ with only one λ: either $\lambda x_1 \ldots x_n.\ M$, or even shorter $\lambda \vec{x}.\ M$. 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 write $M[x_1 := N_1, \ldots, x_n := N_n]$ for the term obtained from M by the simultaneous capture-avoiding substitution of $N_1, \ldots, N_n$ for the variables $x_1, \ldots, x_n$. All the substitutions we shall consider map variables to terms of the same type. When working with an abstract substitution σ, we write $M.\sigma$ for the term obtained by applying σ to M. We use the usual operational semantics of the calculus,

[1] We use a unique atomic type, but our approach generalizes without problems to any number of atomic types.

$\beta\delta$-reduction ($\xrightarrow{*}_{\beta\delta}$) which is the reflexive transitive closure of the union of the relations of β-contraction ($\to_\beta$) and δ-contraction ($\to_\delta$) which are the following rewriting relations:

$$(\lambda x.M)N \to_\beta M[x := N] \quad YM \to_\delta M(YM)$$

Definition 2.1. The *Böhm tree* of a term M is a possibly infinite labeled tree that is defined co-inductively. If M can be reduced so as to obtain a term of the form $\lambda\vec{x}.N_0N_1\dots N_k$ with N_0 a variable or a constant, then $BT(M)$ is a tree whose root is labeled by $\lambda\vec{x}.N_0$ and the immediate successors of its root are $BT(N_1)$, …, $BT(N_k)$. Otherwise $BT(M)$ is a single node tree whose root is labeled Ω^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 2.2. 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 $BT((YF.N)a) = aba^2ba^4b\dots a^{2^n}b\dots$ after every consecutive occurrence of b the number of occurrences of a doubles because of the double application of g inside N.

wMSOL 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* works on trees over a fixed tree signature Σ. It is a tuple:

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

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 $(S_1, \dots, S_i)$ 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, i.e. $\rho(q') \le \rho(q)$.

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

Automata will work on Σ-labeled trees that are partial functions $t : \mathbb{N}^* \dot{\to} \Sigma \cup \{\Omega\}$ whose domain of definition satisfies the usual requirements, and 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 (i.e. $\delta_0(q, c)$ is not empty). If v 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 $(S_1, \dots, S_i) \in \delta(q, t(v))$; then Adam chooses S_j (for $j = 1, \dots, i$) 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 [19], are obtained by requiring that all states have rank 0. Automata with co-trivial conditions are just the ones where 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, it consists of terms whose Böhm trees are accepted by the automaton from its initial state q^0:

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

Observe that $L(\mathcal{A})$ is closed under $\beta\delta$-conversion since the Church-Rosser property of the calculus implies that two $\beta\delta$-convertible terms have the same Böhm tree.

Example 2.3. Consider a weak alternating automaton $\mathcal{A}$ defining the property "action b appears infinitely often". The automaton has states $Q = \{q_1, q_2\}$, and the signature $\Sigma = \{a, b\}$ consisting of two constants of type $o \to o$. Over this signature, the Böhm trees are just sequences. The transitions of $\mathcal{A}$ are:

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

The ranks of states are indicated by their subscripts. When started in q_2 the automaton spawns a run from q_1 each time it sees letter a. The spawned runs must stop in order to accept, and they stop when they see letter b (cf. Figure 2). So every a must be eventually followed by b.

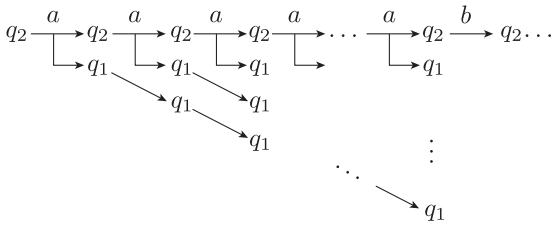

Figure 2: A run of an alternating automaton on $a \ldots ab \ldots$

Models. We use standard notions and notations for models for λY-calculus, in particular for *valuation/variable assignment* and of *interpretation of a term* (see [16]). We shall write $[\![M]\!]^{\mathcal{S}}_{\nu}$ for the interpretation of a term M in a model $\mathcal{S}$ with the valuation ν. As usual, we will omit subscripts or superscripts in the notation of the semantic function if they are clear from the context.

The simplest models of λY-calculus are based on monotone functions. A *GFP-model* of a signature Σ is a tuple $\mathcal{S} = \langle \{\mathcal{S}_A\}_{A \in \mathcal{T}}, \rho \rangle$ where $\mathcal{S}_o$ is a finite lattice, called the *base set* of

the model, and for every type $A \to B \in \mathcal{T}$, $\mathcal{S}_{A\to B}$ is the lattice $\mathrm{mon}[\mathcal{S}_A \mapsto \mathcal{S}_B]$ of monotone functions from $\mathcal{S}_A$ to $\mathcal{S}_B$ ordered coordinate-wise. The valuation function ρ is required to interpret Ω^A as the greatest element of $\mathcal{S}_A$, and Y^A as the greatest fixpoint operator of functions in $\mathcal{S}_{A\to A}$. Observe that every $\mathcal{S}_A$ is finite, hence all the greatest fixpoints exist without any additional assumptions on the lattice.

We can now adapt the definition of recognizability by semigroups taken from language theory to our richer models.

Definition 2.4. A GFP model $\mathcal{S}$ over the base set $\mathcal{S}_o$ *recognizes a language* L of closed λY-terms of type o if there is a subset $F \subseteq \mathcal{S}_o$ such that $L = \{M \mid [\![M]\!]^{\mathcal{S}} \in F\}$.

A direct consequence of Statman's finite completeness theorem [35] is that such models can characterize a term up to equality: $BT(M) = BT(N)$ iff the values of M and N are the same in every monotone models. This property is sufficient for our purposes. The celebrated result of Loader [24] implies that we cannot hope for a much stronger completeness property, and have good algorithmic qualities at the same time.

The following theorem characterizes the recognizing power of GFP models.

Theorem 2.5 ([34])**.** *A language L of λY-terms is recognized by a GFP-model iff it is a boolean combination of languages recognized by weak automata whose all states have rank* 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 wMSOL properties of Böhm trees of terms. We will rely on the equivalence of wMSOL 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 3.2).

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

The type system we propose is obtained by allowing the use of intersections inside simple types. This idea has been used by Kobayashi [20] 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 $\mathrm{Types}^k_A = \bigcup_{0\le l\le k} \mathrm{types}^l_A$ 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. This technical choice amounts to quotient types with respect to the associativity, the commutativity and the idempotency of the intersection operator. A nice consequence in our context is that the sets types^k_A and Types^k_A are finite.

Notice also that the application of the intersection type operator to two sets of types is then represented by the union of those two sets.

Example 3.1. Suppose that in Q we have states q_0, r_0, q_1 with ranks given by their subscripts. A type $\{q_0, r_0\} \subseteq \mathrm{Types}^0_o$ will type terms whose Böhm tree is accepted both from q_0 and from r_0. A type $\{\{q_0, r_0\} \to \{q_1\}\} \subseteq \mathrm{types}^1_{o\to o}$ will type terms that when given a term of type $\{q_0, r_0\}$ produce a Böhm tree accepted from q_1. Observe that, for example, $\{\{q_0, r_0\} \to \{q_1\}, \{q_1\} \to \{q_1\}\} \subseteq \mathrm{Types}^1_{o\to o}$ while $\{\{q_1\}, \{q_1\} \to \{q_1\}\}$ is not an intersection type in our sense since the two types in the set have different underlying simple types.

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 (the first rule can be derived from the second and third one, but we find it more intuitive to explicitly spell it out). Usually in intersection types, the lower a type is in the subsumption order, the less terms it can type. Here we take the information order instead, the lower a type is in the subsumption order, the less precise it is. As we said earlier, this choice is motivated by the connection of types with models, the information order gives us an isomorphism between the two, while the usual choice would give us a duality between types and models.

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

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

The typing system presented in Figure 3 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 $S \subseteq \mathrm{Types}_A$ 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 $S_i \subseteq \mathrm{Types}_{A_i}$. We will use a functional notation and write $\Gamma(x_i)$ for S_i. We shall also write $\Gamma, x \geq S$ for an extension of the environment Γ with the declaration $x \geq S$.

The rules in the first row of Figure 3 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 with the caveat that the abstraction rule incorporates the stratification of types with respect to ranks so that the types used in the judgment are always well-formed. The least fixpoint rule in the next line is standard, it expresses that the least fixpoint can be approximated by iterations started in the least element: if we derive $\Gamma \vdash \lambda x.M \geq \emptyset \to T$ then we obtain that $\Gamma \vdash Yx.M \geq T$, that can allow us to derive $\Gamma \vdash Yx.M \geq T'$ provided $\Gamma \vdash \lambda x.M \geq T \to T'$ etc. 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.

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

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

$$\frac{}{\Gamma \vdash c \geq \{q : \delta_0(q, c) \neq \emptyset\}} \qquad \frac{(S_1, \ldots, S_i) \in \delta_i(a, q)}{\Gamma \vdash a \geq \{S_1 \to \cdots \to S_i \to q\}}$$

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

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

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

Figure 3: Type system

Theorem 3.2. *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 [21] and to Tsukada and Ong type system [38], the fixpoint typing rules we propose do not refer to an external parity game. Our type system does not require *flagged types*, and is on the contrary based on a standard treatment of free variables. In the example below we use fixpoint rules on terms of order 2.

In order to prove Theorem 3.2 we will need to formulate and prove a more general statement that concerns terms of all types (Theorem 5.5). To describe the properties of the type system in higher types, we will construct a model from our fixed automaton $\mathcal{A}$, and show (Theorem 4.13) that the model recognizes $L(\mathcal{A})$ in the sense of Definition 2.4. Then Theorem 5.5 will say that the type system reflects the values of the terms in the model.

Due to the symmetries in weak alternating automata, and in the model we are going to construct, we will obtain also a dual type system. This system can be used to show that the Böhm tree of a term is not accepted by the automaton.

The dual type system is presented in Figure 4 on page 10. The notation is as before but we now define $S(T)$ to be $\{s : U \to s \in S \wedge U \sqsupseteq T\}$. The rules for application, abstraction and variable do not change. By duality we obtain:

Corollary 3.3. *For every closed term M of type o and every state q of $\mathcal{A}$: judgment $\vdash M \not\geq q$ is derivable iff $\mathcal{A}$ does not accept $BT(M)$ from q.*

So the two type systems together allow us to derive both positive and negative information about a program.

We finish this section with two moderate size examples of typing derivations.

Example 3.4. Take a signature consisting of two constants $c : o$ and $a : o \to o$. We consider an extremely simple weak alternating automaton with just one state q of rank 1 and transitions:

$$\delta(q, a) = \{q\} \qquad \delta(q, c) = \emptyset\ .$$

$$\frac{S \subseteq T \subseteq Q}{S \sqsupseteq_o T} \qquad \frac{\forall s \in S, \exists t \in T, s \sqsupseteq_A t}{S \sqsupseteq_A T} \qquad \frac{s = t}{s \sqsupseteq_o t} \qquad \frac{T \sqsupseteq_A S \quad s \sqsupseteq_B t}{S \to s \sqsupseteq_{A\to B} T \to t}$$

$$\frac{}{\Gamma \vdash c \geq \{q : \delta_o(q,c) = \emptyset\}} \qquad \frac{\forall (S_1,S_2) \in \delta(a,q), (T_1 \cap S_1) \cup (T_2 \cap S_2) \neq \emptyset}{\Gamma \vdash a \not\geq T_1 \to T_2 \to q}$$

$$\frac{\Gamma \vdash M \not\geq S \quad \Gamma \vdash N \not\geq T}{\Gamma \vdash MN \not\geq S(T)} \qquad \frac{S \in \mathrm{Types}^k,\ T \subseteq \mathrm{types}^k \quad \Gamma, x \not\geq S \vdash M \not\geq T}{\Gamma \vdash \lambda x.M \not\geq S \to T}$$

$$\frac{S \subseteq \mathrm{types}_A^{2k+1}, \quad T \in \mathrm{Types}_A^{2k}, \quad \Gamma \vdash \lambda x.M \not\geq (S \cup T) \to S \quad \Gamma \vdash Yx.M \not\geq T}{\Gamma \vdash Yx.M \not\geq S \cup T}\ Y \text{ odd}$$

$$\frac{\Gamma \vdash (\lambda x.M) \not\geq S \quad \Gamma \vdash (Yx.M) \not\geq T}{\Gamma \vdash Yx.M \not\geq S(T)}\ Y \text{ even}$$

Figure 4: Dual type system

This automaton accepts the finite sequences of a's ending in c. Observe that these transition rules give us typing axioms

$$\frac{}{\vdash a \geq \{q\} \to q} \qquad \frac{}{\vdash c \geq q}$$

Notice that we omit some set parenthesis over singletons; so for example we write $c \geq q$ instead of $c \geq \{q\}$. In this example we will still keep the parenthesis to the left of the arrow to emphasize that we are in our type system, and not in simple types. In the next example we will omit them too.

First, let us look at a term $G_1 \equiv \lambda f^{o\to o} \lambda x^o.\ f(fx)$. It has a simple type $\tau_1 \equiv (o \to o) \to o \to o$. The judgment $\vdash G_1 \geq \gamma_1$ is derivable in our system; where $\gamma_1 \equiv \{\{q\} \to q\} \to \{q\} \to q$.

$$\dfrac{\dfrac{\Gamma \vdash f \geq \{q\} \to q \quad \dfrac{\Gamma \vdash f \geq \{q\} \to q \quad \Gamma \vdash x \geq q}{\Gamma \vdash fx \geq q}}{\Gamma \vdash f(fx) \geq q}}{\vdash \lambda f \lambda x.\ f(fx) \geq \gamma_1}$$

here $\Gamma \equiv f \geq \{q\} \to q,\ x \geq q$.

Consider now $G_2 \equiv \lambda f^{\tau_1} \lambda x^{o\to o}.\ f(fx)$. We have that G_2 is of type $\tau_2 \equiv \tau_1 \to \tau_1$. A derivation very similar to the above will show $\vdash G_2 \geq \gamma_2$ where $\gamma_2 \equiv \{\gamma_1\} \to \gamma_1$.

Then by induction we can define $G_i \equiv \lambda f^{\tau_{i-1}} \lambda x^{\tau_{i-2}}.\ f(fx)$ that is of a type $\tau_i \equiv \tau_{i-1} \to \tau_{i-1}$. Still a similar derivation as above will show $\vdash G_i \geq \gamma_i$ where $\gamma_i \equiv \{\gamma_{i-1}\} \to \gamma_{i-1}$.

One use of application rule then shows that $G_i G_{i-1} \geq \gamma_{i-1}$:

$$\frac{\vdash G_i \geq \{\gamma_{i-1}\} \to \gamma_{i-1} \quad \vdash G_{i-1} \geq \gamma_{i-1}}{\vdash G_i G_{i-1} \geq \gamma_{i-1}}$$

In consequence, we can construct by induction a derivation of

$$\vdash G_i G_{i-1} \dots G_1 ac \geq q$$

This derivation proves that the Böhm tree of $G_i G_{i-1} \dots G_1 ac$ is a sequence of a's ending in c. While the length of this sequence is a tower of exponentials in the height i, the typing derivation we have constructed is linear in i (if types are represented succinctly). This simple

example, already analyzed in [20], shows the power of modular reasoning provided by the typing approach. We should note though that if the initial automaton had two states, the number of potential types would also be roughly the tower of exponentials in i. Due to the complexity bounds [36], there are terms and automata for which there is no small derivation. Yet one can hope that in many cases a small derivation exists. For example, if we wanted to show that the length of the sequence is even then the automaton would have two states but the derivation would be essentially the same.

Example 3.5. 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 5, $BT(M) = aba^2ba^4b \dots a^{2^n}b \dots$. We show with typing that there are infinitely many occurrences of b in $BT(M)$. To this end we take an automaton having states $Q = \{q_1, q_2\}$, and working over the signature containing 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:

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

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 the 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:

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

We have displayed only one of the two premises of the $Yodd$ 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.

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

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 proved directly using application and intersection rules.

4. Models for weak automata

We describe a construction of a model that recognizes, in a sense of Definition 2.4, the language defined by a weak automaton. The 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. We shall work with (finite) complete lattices and with monotone functions between complete lattices. In the first subsection we define the basic structure of the model. Its properties will allow us later to define fixpoints and show that indeed we can interpret the λY-calculus in the model. In the last subsection we will show that with an appropriate interpretation of constants the model can recognize the language of a given weak automaton.

The challenge in this construction comes from the fact that using only the least or only the greatest fixpoints is not sufficient. Indeed, we have shown in [34] 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 5). 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 6.

4.1. **A stratified model.** We fix a finite set of states Q and a ranking function $\rho : Q \to \mathbb{N}$. 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 $Q_{\leq k} = \{q \in Q : \rho(q) \leq k\}$.

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 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

$$\mathcal{D}^0_o = \mathcal{P}(Q_0) \quad \text{and} \quad \mathcal{D}^0_{A\to B} = \mathrm{mon}[\mathcal{D}^0_A \mapsto \mathcal{D}^0_B].$$

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 structure 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: for all f in $\mathcal{D}^k_{A\to B}$ and g in $\mathcal{D}^k_A$, $f(g)$ is in $\mathcal{D}^k_B$. 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 $\mathcal{D}^k_A$. We review this structure here, and provide necessary justification in lemmas that follow. The first thing to notice is that for each type A, the set $\mathcal{D}^k_A$ 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 5 represents schematically these essential properties of $\mathcal{D}^k_A$. Notice that, in Figure 5, for every element e' in $\mathcal{L}^k(e)$ (or d' in $\mathcal{L}^k(d)$) we have $e'^{\downarrow} = e$ (or $d'^{\downarrow} = d$). All these properties are consequences of the following lemma and of Lemma 4.5.

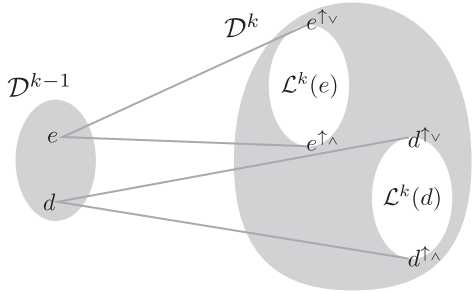

Figure 5: Relation between models $\mathcal{D}^{k-1}$ and $\mathcal{D}^k$. Every element in $\mathcal{D}^{k-1}$ is related to a sub-lattice of elements in $\mathcal{D}^k$.

Lemma 4.1. *For every $0 < k \leq m$, and every type A, we have:*

(0) *$\mathcal{L}^k_A$ is a lattice: if (d_1, d_2), (e_1, e_2) are in $\mathcal{L}^k_A$, then so are $(d_1 \vee e_1, d_2 \vee e_2)$ and $(d_1 \wedge e_1, d_2 \wedge e_2)$.*

(1) *Given (d_1, d_2) and (e_1, e_2) in $\mathcal{L}^k_A$, if $d_2 \leq e_2$ then $d_1 \leq e_1$.*

(2) *Given d_2 in $\mathcal{D}^k_A$, there is a unique d_1 in $\mathcal{D}^{k-1}_A$, so that (d_1, d_2) in $\mathcal{L}^k_A$. Let us denote this unique element $d_2^{\downarrow}$.*

(3) *If $d_1 \leq e_1$ in $\mathcal{D}^{k-1}_A$, then:*
   (a) *there is $e_1^{\uparrow\vee}$ in $\mathcal{L}^k_A(e_1)$ so that for every d_2 in $\mathcal{L}^k_A(d_1)$ we have $d_2 \leq e_1^{\uparrow\vee}$,*
   (b) *there is $d_1^{\uparrow\wedge}$ in $\mathcal{L}^k_A(d_1)$ so that for every e_2 in $\mathcal{L}^k_A(e_1)$ we have $d_2^{\uparrow\wedge} \leq e_2$.*

Proof. Item 0 can be proved by a straightforward induction on the size of types.

The proof of items $1, 2, 3$, is by simultaneous induction on the size of A. Notice that item 2 is an immediate consequence of item 1. We shall therefore not prove it, but we feel free to use it as an induction hypothesis.

We start with the case when A is the base type o:

Ad 1. In that case $\mathcal{D}^k_A = Q_{\leq k}$, $e_1 = e_2 \cap Q_{\leq k-1}$ and $d_1 = d_2 \cap Q_{\leq k-1}$. Thus, we indeed have that $d_2 \leq e_2$ implies that $d_1 \leq e_1$.

Ad 3. Here, we have $\mathcal{D}^{k-1}_A = Q_{\leq k-1}$ and then letting $e_1^{\uparrow\vee} = e_1 \cup Q_k$ and $d_2^{\uparrow\wedge} = d_2$ is enough to conclude.

Let us now suppose that $A = B \to C$:

Ad 1. Given f_1 in $\mathcal{D}^{k-1}_B$, by induction hypothesis, using item 3, we know that there exist f_2 in $\mathcal{D}^k_B$ so that (f_1, f_2) is in $\mathcal{L}^k_B$. Thus, we have $(d_1(f_1), d_2(f_2))$ and $(e_1(f_1), e_2(f_2))$ in $\mathcal{L}_{k,C}$. With the assumption that $d_2 \leq e_2$, we obtain $d_2(f_2) \leq e_2(f_2)$. By induction hypothesis we get $e_1(f_1) \leq d_1(f_1)$. As f_1 is arbitrary, we can conclude that $e_1 \leq d_1$.

Ad 3. By induction hypothesis, using item 2, for f_2 in $\mathcal{D}^k_B$, there is a unique element $f_2^{\downarrow}$ of $\mathcal{D}^{k-1}_B$ so that $(f_2^{\downarrow}, f_2)$ is in $\mathcal{L}^k_B$. Given h_1 in $\mathcal{D}^{k-1}_A$ we define for every element f_2 in $\mathcal{D}^k_B$:

$$h_1^{\uparrow\vee}(f_2) = (h_1(f_2^{\downarrow}))^{\uparrow\vee} \qquad h_1^{\uparrow\wedge}(f_2) = (h_1(f_2^{\downarrow}))^{\uparrow\wedge} \ .$$

We will verify only item 3(a), the case of $h_1^{\uparrow\wedge}$ being analogous.

We need to check that $h_1^{\uparrow\vee}$ is in $\mathcal{D}^k_A$. First of all we need to check that it is in $\mathrm{mon}[\mathcal{D}^k_B \mapsto \mathcal{D}^k_C]$. Take g_2 and f_2 in $\mathcal{D}^k_B$ so that $g_2 \leq f_2$. By induction hypothesis, using item 1, we have that $g_2^{\downarrow} \leq f_2^{\downarrow}$. Then $h_1(g_2^{\downarrow}) \leq h_1(f_2^{\downarrow})$ by monotonicity of h_1. From item 3 of induction hypothesis we obtain $(h_1(g_2^{\downarrow}))^{\uparrow\vee} \leq (h_1(f_2^{\downarrow}))^{\uparrow\vee}$; proving that $h_1^{\uparrow\vee}$ is monotone.

Next, we show that $(h_1, h_1^{\uparrow\vee})$ is in $\mathcal{L}^k_A$. If we take $(f_2^{\downarrow}, f_2)$ in $\mathcal{L}_{k,B}$, we obtain by the induction hypothesis, item 3, that $(h_1(f_2^{\downarrow}), (h_1(f_2^{\downarrow}))^{\uparrow\vee})$ is in $\mathcal{L}^k_C$. But, by our definition, this implies that $(h_1(f_2^{\downarrow}), h_1^{\uparrow\vee}(f_2))$ is in $\mathcal{L}^k_C$. As f_2 is arbitrary we obtain that $(h_1, h_1^{\uparrow\vee})$ is indeed in $\mathcal{L}^k_A$ proving then that $h_1^{\uparrow\vee}$ is in $\mathcal{D}^k_A$.

It remains to prove that, given $d_1 \leq e_1$ in $\mathcal{D}^k_A$, for all d_2 in $\mathcal{L}^k_A(d_1)$ we have $d_2 \leq e_1^{\uparrow\vee}$. Given $(f_2^{\downarrow}, f_2)$ in $\mathcal{L}^k_B$, we have $(d_1(f_2^{\downarrow}), d_2(f_2))$ in $\mathcal{L}^k_C$. Using induction hypothesis, item 3, we get $d_2(f_2) \leq (d_1(f_2^{\downarrow}))^{\uparrow\vee}$. Since $d_1 \leq e_1$ we obtain $d_2(f_2) \leq (e_1(f_2^{\downarrow}))^{\uparrow\vee}$ that is the desired $d_2(f_2) \leq e_1^{\uparrow\vee}(f_2)$. As f_2 is arbitrary, this shows that $d_2 \leq e_1^{\uparrow\vee}$. □

Notice that the fact that $\mathcal{D}^k_A$ is a complete lattice is an immediate consequence from the fact that $\mathcal{L}^k_A$ is a complete lattice.

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 $\mathcal{D}^k_A$.

Corollary 4.2. *For each $k \leq m$, and each type A, $\mathcal{D}^k_A$ is a non-empty lattice.*

The mappings $(\cdot)^{\downarrow}$ and $(\cdot)^{\uparrow\vee}$ form a Galois connection between $\mathcal{D}^k_A$ and $\mathcal{D}^{k-1}_A$. For every $d_2 \in \mathcal{D}^k_A$ and $e_1 \in \mathcal{D}^{k-1}_A$ we have: $d_2^{\downarrow} \leq e_1$ iff $d_2 \leq e_1^{\uparrow\vee}$.

Similarly $(\cdot)^{\downarrow}$ and $(\cdot)^{\uparrow\wedge}$ form a Galois connection between $\mathcal{D}^{k-1}_A$ and $\mathcal{D}^k_A$. For every $d_1 \in \mathcal{D}^{k-1}_A$ and $e_2 \in \mathcal{D}^k_A$ we have: $d_1^{\uparrow\wedge} \leq e_2$ iff $d_1 \leq e_2^{\downarrow}$.

From now on, $\bot^k_A$ and $\top^k_A$ will denote the least and the greatest element of $\mathcal{D}^k_A$.

Corollary 4.3. *Given d_2 in $\mathcal{D}^k_{B\to C}$ and e_2 in $\mathcal{D}^k_B$, we have $(d_2(e_2))^{\downarrow} = d_2^{\downarrow}(e_2^{\downarrow})$. Given d_1 in $\mathcal{D}^{k-1}_{B\to C}$ and e_2 in $\mathcal{D}^k_B$, we have: $d_1^{\uparrow\wedge}(e_2) = (d_1(e_2^{\downarrow}))^{\uparrow\wedge}$ and $d_1^{\uparrow\vee}(e_2) = (d_1(e_2^{\downarrow}))^{\uparrow\vee}$.*

Finally, we show one more decomposition property of our models that we will use later to show a correspondence between types and elements of the model (Lemma 5.2).

Definition 4.4. We define an operation $\overline{d}$ on the elements d of $\mathcal{D}^k_A$ by induction on A:

- if $A = o$, then $\overline{d} = d \cap Q_k$,
- if $A = B \to C$, then for e in $\mathcal{D}^k_B$, $\overline{d}(e) = \overline{d(e)}$.

A simple induction shows that the operation $\overline{(\cdot)}$ is an involution and that $\overline{\overline{d}} = \overline{d}$. A similar induction shows that $\overline{(\cdot)}$ is the identity function on $\mathcal{L}^k_A(\bot^{k-1}_A)$.

Lemma 4.5. *For every $0 < k \leq m$, and every d in $\mathcal{D}^k_A$; let $\overline{d}$ be as defined above, and let $\widetilde{d} = (\top^{k-1}_A)^{\uparrow\wedge} \vee \overline{d}$. We have $d = ((d^{\downarrow})^{\uparrow\wedge}) \vee \overline{d}$ and $d = ((d^{\downarrow})^{\uparrow\vee}) \wedge \widetilde{d}$.*

Proof. We prove only the first identity, the second being similar. The proof is by induction on the size of A.

In case $A = o$, from definitions we get $\overline{d} = d \cap Q_k$ while $(d^{\downarrow})^{\uparrow\wedge} = d \cap Q_{\leq k-1}$. Thus we indeed have $d = ((d^{\downarrow})^{\uparrow\wedge}) \vee \overline{d}$.

In case $A = B \to C$. Given e in $\mathcal{D}^k_B$, we have that $(d^{\downarrow})^{\uparrow\wedge}(e) = (d^{\downarrow}(e^{\downarrow}))^{\uparrow\wedge} = ((d(e))^{\downarrow})^{\uparrow\wedge}$. Moreover, $\overline{d}(e) = \overline{d(e)}$. Therefore, $(((d^{\downarrow})^{\uparrow\wedge}) \vee \overline{d})(e) = (((d(e))^{\downarrow})^{\uparrow\wedge}) \vee \overline{d(e)}$. But, by induction, we have $d(e) = (((d(e))^{\downarrow})^{\uparrow\wedge}) \vee \overline{d(e)} = (((d^{\downarrow})^{\uparrow\wedge}) \vee \overline{d})(e)$. As e is arbitrary, we get the identity. □

Lemma 4.6. *Given e_1, e_2 in $\mathcal{D}^k_A$ and d in $\mathcal{D}^{k-1}_A$:*

- *if $\overline{e_1} \neq \overline{e_2}$ then $d^{\uparrow\wedge} \vee \overline{e_1} \neq d^{\uparrow\wedge} \vee \overline{e_2}$,*
- *if $\widetilde{e_1} \neq \widetilde{e_2}$ then $d^{\uparrow\vee} \wedge \overline{e_1} \neq d^{\uparrow\vee} \wedge \overline{e_2}$,*

Proof. We only prove the first identity, the second being essentially dual.

We proceed by induction on A. When $A = o$, then as $d^{\uparrow\wedge} = d$ (see the proof of Lemma 4.1), $d \subseteq Q_{\leq k-1}$ and $\overline{e_1}$, $\overline{e_2}$ are included in Q_k, the conclusion is immediate.

When $A = B \to C$, since $\overline{e_1} \neq \overline{e_2}$ there is a so that $\overline{e_1}(a) \neq \overline{e_2}(a)$. Now we have $(d^{\uparrow\wedge} \vee \overline{e_i})(a) = d^{\uparrow\wedge}(a) \vee \overline{e_i}(a) = (d(a^{\downarrow}))^{\uparrow\wedge} \vee \overline{e_i(a)}$ for i in $\{1, 2\}$. The induction hypothesis implies that $(d(a^{\downarrow}))^{\uparrow\wedge} \vee \overline{e_1(a)} \neq (d(a^{\downarrow}))^{\uparrow\wedge} \vee \overline{e_2(a)}$. Therefore $(d^{\uparrow\wedge} \vee \overline{e_1})(a) \neq (d^{\uparrow\wedge} \vee \overline{e_2})(a)$ and $d^{\uparrow\wedge} \vee \overline{e_1} \neq d^{\uparrow\wedge} \vee \overline{e_2}$ as desired. □

Lemma 4.5 shows that every element d_2 in $\mathcal{L}^k_A(d_1)$ is of the form $d_1^{\uparrow\wedge} \vee e_2$ with e_2 in $\mathcal{L}^k_A(\bot^{k-1}_A)$ and of the form $d_1^{\uparrow\vee} \wedge e_2$ with e_2 in $\mathcal{L}^k_A(\top^{k-1}_A)$. Thus not only $\mathcal{L}^k_A$ puts every element d_1 in relation with a lattice $\mathcal{L}^k(d_1)$, but, from Lemma 4.6, this relation is injective. As a consequence the lattice $\mathcal{L}^k_A(d_1)$ is isomorphic to $\mathcal{L}^k_A(\bot^{k-1}_A)$ and to $\mathcal{L}^k_A(\top^{k-1}_A)$, showing as we said earlier, that for any e and d in $\mathcal{D}^{k-1}_A$, $\mathcal{L}^k_A(e)$ is isomorphic to $\mathcal{L}^k_A(d)$. Another consequence is that $\mathcal{D}^k_A$ is isomorphic to the lattice $\mathcal{D}^{k-1}_A \times \mathcal{L}_{k,A}(\bot^{k-1}_A)$ or to the lattice $\mathcal{D}^{k-1}_A \times \mathcal{L}_{k,A}(\top^{k-1}_A)$. This isomorphism is important as it shows that the types we have described Section 3 are a faithful representation of the elements of the model we have just constructed.

4.2. **Fixpoints in a stratified model.** The properties from the previous subsection allow us to define fixpoint operators in every applicative structure $\mathcal{D}^k$. Then we can show that a stratified structure is a model of λY-calculus.

Definition 4.7. 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 6. On stratum 0 it is just the greatest fixpoint. Then this greatest fixpoint, call it d, is lifted to stratum 1, and the least fixpoint computation is started in the complete sub-lattice of the refinements of d. 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.

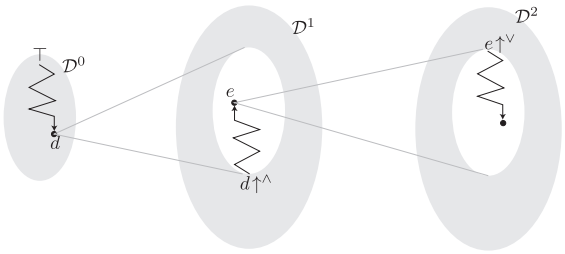

Figure 6: A computation of a fixpoint: it starts in $\mathcal{D}^0$, and then the least and the greatest fixpoints alternate.

It remains to show that equipped with the interpretation of fixpoints given by Definition 4.7 the applicative structure $\mathcal{D}^k$ is a model of the λY-calculus. First, we check that fix^k_A is indeed an element of the model and that it is a fixpoint.

Lemma 4.8. *For every $0 \leq k \leq m$ and every type A we have that fix^k_A is monotone, and if $k > 0$ then $(\mathrm{fix}^{k-1}_A, \mathrm{fix}^k_A) \in \mathcal{L}^k_{(A\to A)\to A}$. Moreover for every f in $\mathcal{D}^k_{A\to A}$, $f(\mathrm{fix}^k_A(f)) = \mathrm{fix}^k_A(f)$.*

Proof. We proceed by induction on k. For 0 the statement is obvious. We will only consider the case where k is even, the other being dual.

Consider the case $2k > 0$. First we show monotonicity. Suppose $g \leq h$ are two elements of $\mathcal{D}^{2k}_{A\to A}$. By Lemma 4.1 we get $g^{\downarrow} \leq h^{\downarrow}$. Consider $g^1 = \mathrm{fix}^{2k-1}_A(g^{\downarrow})$ and $h^1 = \mathrm{fix}^{2k-1}_A(h^{\downarrow})$. By induction hypothesis fix^{2k-1}_A is monotone, so $g^1 \leq h^1$. Then, once again using the Lemma 4.1, we have $(g^1)^{\uparrow\vee} \leq (h^1)^{\uparrow\vee}$. This implies $\mathrm{fix}^{2k}_A(g) \leq \mathrm{fix}^{2k}_A(h)$.

Now we show $(\mathrm{fix}^{2k-1}_A, \mathrm{fix}^{2k}_A) \in \mathcal{L}^{2k}_{(A\to A)\to A}$. We take an arbitrary pair $(f_1, f_2) \in \mathcal{L}^{2k}_{A\to A}$ and we need to show that $(\mathrm{fix}^{2k-1}_A(f_1), \mathrm{fix}^{2k}_A(f_2)) \in \mathcal{L}^{2k}_A$. This follows from the following

calculation

$$\begin{aligned}(\mathrm{fix}^{2k-1}(f_1), (\mathrm{fix}^{2k-1}(f_1))^{\uparrow\vee}) &\in \mathcal{L}_A^{2k} && \text{by Lemma 4.1}\\ (f_1(\mathrm{fix}^{2k-1}(f_1)), f_2((\mathrm{fix}^{2k-1}(f_1)^{\uparrow\vee})) &\in \mathcal{L}_A^{2k} && \text{by logical relation}\\ (\mathrm{fix}^{2k-1}(f_1), f_2((\mathrm{fix}^{2k-1}(f_1)^{\uparrow\vee})) &\in \mathcal{L}_A^{2k} && \text{since } \mathrm{fix}^{2k-1}(f_1)\\ & && \text{is a fixpoint of } f_1\\ (\mathrm{fix}^{2k-1}(f_1), f_2^i((\mathrm{fix}^{2k-1}(f_1)^{\uparrow\vee})) &\in \mathcal{L}_A^{2k} && \text{for every } i \geq 0\end{aligned}$$

Moreover, from Lemma 4.1, we have that $f_2((\mathrm{fix}^{2k-1}(f_1)^{\uparrow\vee}) \leq (\mathrm{fix}^{2k-1}(f_1))^{\uparrow\vee}$. This implies

$$f_2^{i+1}((\mathrm{fix}^{2k-1}(f_1)^{\uparrow\vee}) \leq f_2^i((\mathrm{fix}^{2k-1}(f_1))^{\uparrow\vee})$$

for every $i \in \mathbb{N}$. Therefore, $(f_2^i((\mathrm{fix}^{2k-1}(f_1))^{\uparrow\vee})_{i\in\mathbb{N}}$ is a decreasing sequence of $\mathcal{D}_A^{2k}$. Since the model is finite this sequence reaches the fixpoint, namely $\mathrm{fix}_A^{2k}(f_2) = f_2^i((\mathrm{fix}_A^{2k-1}(f_1)^{\uparrow\vee}))$ for some i. Thus, at the same time, this shows that $(\mathrm{fix}_A^{2k-1}(f_1), \mathrm{fix}_A^{2k}(f_2)) \in \mathcal{L}_A^{2k}$ and that $\mathrm{fix}^{2k}(f_2)$ is a fixpoint of f_2. □

This lemma has the following interesting corollary that will prove useful in the study of type systems.

Corollary 4.9. *For $k > 0$, A a type, and $f \in \mathcal{D}^k_{A\to A}$ we have*

$$\mathrm{fix}_A^{2k}(f) = \bigvee\{d \mid f(d) \geq d \text{ and } d^{\downarrow} = \mathrm{fix}_A^{2k-1}(f)\}$$
$$\mathrm{fix}_A^{2k+1}(f) = \bigwedge\{d \mid f(d) \leq d \text{ and } d^{\downarrow} = \mathrm{fix}_A^{2k}(f)\}$$

Moreover, the fundamental lemma on logical relations gives us the following consequence.

Corollary 4.10. *For every $k > 0$, every term M and valuation ν into $\mathcal{D}^{k-1}$ we have $([\![M]\!]^{k-1}_{\nu}, [\![M]\!]^k_{\nu^{\uparrow\wedge}}) \in \mathcal{L}^k$ and $([\![M]\!]^{k-1}_{\nu}, [\![M]\!]^k_{\nu^{\uparrow\vee}}) \in \mathcal{L}^k$; where $\nu^{\uparrow\wedge}$ and $\nu^{\uparrow\vee}$ are as expected, i.e. $\nu^{\uparrow\wedge}(x) = (\nu(x))^{\uparrow\wedge}$ and $\nu^{\uparrow\vee}(x) = (\nu(x))^{\uparrow\vee}$.*

Now we turn to showing that for every k, $\mathcal{D}^k$ is indeed a model of λY-calculus. Since $\mathcal{D}^k_{A\to B}$ does not contain all the functions from $\mathcal{D}^k_A$ to $\mathcal{D}^k_B$ we must show that there are enough of them to form a model of λY, the main problem being to show that $[\![\lambda x.M]\!]^{\mathcal{D}^k}_{\upsilon}$ defines an element of $\mathcal{D}^k$. 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 $A_1, \ldots, A_n$ respectively and a term M of type B with free variables in $\vec{x}$, the meaning of M in the model $\mathcal{D}^k$ with respect to $\vec{x}$ will be a function $[\![M]\!]^k_{\vec{x}}$ in $\mathcal{D}^k_{A_1} \to \cdots \to \mathcal{D}^k_{A_n} \to \mathcal{D}^k_B$ that represents the function $\lambda\vec{p}.[\![M]\!]^k_{[p_1/x_1,\ldots,p_n/x_n]}$. Formally it is defined as follows:

(1) $[\![Y^B]\!]^k_{\vec{x}} = \lambda\vec{p}.\ \mathrm{fix}_B$
(2) $[\![a]\!]^k_{\vec{x}} = \lambda\vec{p}.\ \rho(a)$
(3) $[\![x_i^{A_i}]\!]^k_{\vec{x}} = \lambda\vec{p}.p_i$
(4) $[\![MN]\!]^k_{\vec{x}} = \lambda\vec{p}.\ ([\![M]\!]^k_{\vec{x}}\,\vec{p})([\![N]\!]^k_{\vec{x}}\,\vec{p})$
(5) $[\![\lambda y.M]\!]^k_{\vec{x}} = \lambda\vec{p}.\lambda p_y.\ [\![M]\!]^k_{\vec{x}y}\vec{p}p_y$

Note that the 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 parameters $p_1, \ldots, p_n$ ranging respectively in $\mathcal{D}^k_{A_1}, \ldots, \mathcal{D}^k_{A_n}$.

Lemma 4.8 ensures the existence of the meaning of Y in $\mathcal{D}^k$. 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 4.11. *For every sequence of types $\vec{A} = A_1 \ldots A_n$ and every types B, C we have the following:*

- *For every constant $p \in \mathcal{D}^k_B$ the constant function $f_p : A_1 \to \cdots \to A_n \to B$ belongs to $\mathcal{D}^k_{A_1 \to \cdots \to A_n \to B}$.*
- *For $i = 1, \ldots, n$, the projection $\pi_i : A_1 \to \cdots \to A_n \to A_i$ belongs to $\mathcal{D}^k_{A_1 \to \cdots \to A_n \to B}$.*
- *If $f : \vec{A} \to (B \to C)$ and $g : \vec{A} \to B$ are in $\mathcal{D}^k$ then $\lambda \vec{p}. f\vec{p}(g\vec{p}) : \vec{A} \to C$ is in $\mathcal{D}^k$.*

Proof. For the first item we take $p \in \mathcal{D}^k_B$ and show that the constant function $f_p : A \to B$ belongs to $\mathcal{D}^k_{A \to B}$. For $k = 0$ this is clear. For $k > 0$ we take $p^\downarrow$ and consider the constant function $f_{p^\downarrow} \in \mathcal{D}^{k-1}_p$. We have $(f_{p^\downarrow}, f_p) \in \mathcal{L}^k_{A \to B}$ since $(p^\downarrow, p) \in \mathcal{L}^k_B$ by Lemma 4.1. So $f_p \in \mathcal{D}^k_{A \to B}$.

The (easy) proofs for the second and the third items follow the same kind of reasoning. □

These observations allow us to conclude that $\mathcal{D}^k$ is indeed a model of the λY-calculus, that is:

(1) for every term M of type A and every valuation ν ranging of the free variables of M, $[\![M]\!]^k_\nu$ is in $\mathcal{D}^k_A$,
(2) given two terms M and N of type A, if $M =_{\beta\delta} N$, then for every valuation ν, $[\![M]\!]^k_\nu = [\![N]\!]^k_\nu$.

Theorem 4.12. *For every finite set Q, and function $\rho : Q \to \mathbb{N}$. For every $k \leq 0$ the applicative structure $\mathcal{D}^k$ is a model of the λY-calculus.*

4.3. **Correctness and completeness of the model.** We show that the models introduced above are expressive enough to recognize all properties definable by weak alternating automata. For a given automaton we will take a model as defined above, and show that with the right interpretation of constants the model can recognize the set of terms whose Böhm trees are accepted by the automaton (Theorem 4.13).

For the whole section we fix a weak alternating automaton

$$\mathcal{A} = \langle Q, \Sigma, q^0, \delta_o, \delta_{o^2 \to o}, \rho \rangle \ ,$$

where Q is a set of states, Σ is the alphabet, $\delta_o \subseteq Q \times \Sigma$ and $\delta_{o^2 \to o} : Q \times \Sigma \to \mathcal{P}(\mathcal{P}(Q) \times \mathcal{P}(Q))$ are transition functions, and $\rho : Q \to \mathbb{N}$ is a ranking function. For sake of the simplicity of the notation in this section we assume that the only constants in the signature are either of type o or $o^2 \to o$.

Recall that weak means that the states in a transition for a state q have ranks at most $\rho(q)$, in other words, for every $(S_0, S_1) \in \delta(q, a)$, $S_0, S_1 \subseteq Q_{\leq \rho(q)}$. As noted before, without a loss of generality, we assume that δ is monotone, i.e. if $(S_0, S_1) \in \delta(q, a)$ and

$S_0 \subseteq S_0' \subseteq Q_{\leq \rho(q)}$ and $S_1 \subseteq S_1' \subseteq Q_{\leq \rho(q)}$ then $(S_0', S_q') \in \delta(q, a)$. For a closed term M of type o, 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 $BT(M)$.

We want to show that our model $\mathcal{D}^m$ as defined in the previous section can calculate $\mathcal{A}(M)$; here m is the maximal value of the rank function of $\mathcal{A}$. The following theorem states a slightly more general fact. Before proceeding we need to fix the meaning of constants:

$$[\![c]\!]^k = \{q \in Q_{\leq k} : (c, q) \in \delta_o\}$$

$$[\![a]\!]^k(S_0, S_1) = \{q \in Q_{\leq k} : (S_0, S_1) \in \delta_{o^2 \to o}(q, a)\}$$

Notice that, by our assumption about monotonicity of δ, these functions are monotone.

Theorem 4.13. *For every closed term M of type o, and for every $0 \leq k \leq m$ we have: $[\![M]\!]^k = \mathcal{A}(M) \cap Q_{\leq k}$.*

The rest of this section is devoted to the proof of the theorem. For $k = 0$ the model $\mathcal{D}^0$ is just the GFP model over Q_0. Moreover $\mathcal{A}$ restricted to the states in Q_0 is an automaton with trivial acceptance conditions. The theorem follows from Theorem 2.5.

For the induction step consider an even $k > 0$. The case where k is odd is similar and we will not present it here. The two directions of Theorem 4.13 are proved using different techniques. The next lemma shows the left to right inclusion and is based on a rather simple unrolling. The other inclusion is proved using a logical relation between the syntactic model of the λY-calculus and the stratified model (Lemma 4.18). This relation allows us to formally relate the abstractions built into the model to their syntactic meanings that are expressed by the acceptance of Böhm trees of closed λY-terms of atomic type by the weak parity automaton.

Lemma 4.14. $[\![M]\!]^k \subseteq \mathcal{A}(M)$.

Proof. We take $q \in [\![M]\!]^k$ and describe a winning strategy for *Eve* in the acceptance game of $\mathcal{A}$ on $BT(M)$ from q (cf. page 5). If the rank of $q < k$ then such a strategy exists by the induction assumption. So we suppose that $\rho(q) = k$.

If M does not have a head normal form then $BT(M)$ consists just of the root ε labeled with Ω. Then Eve wins by the definition of the game since k is even.

If the head normal form of M is a constant $c : o$ then since $q \in [\![c]\!]^k$ we have $(q, c) \in \delta_o$. Eve wins by the definition of the game.

Suppose then that M has a head normal form aM_0M_1. As $[\![M]\!]^k = [\![aM_0M_1]\!]^k$ we have $q \in [\![aM_0M_1]\!]^k$. By the semantics of a we know that $([\![M_0]\!]^k, [\![M_1]\!]^k) \in \delta(q, a)$. The strategy of Eve is to choose $([\![M_0]\!]^k, [\![M_1]\!]^k)$. Suppose Adam then selects $i \in \{0, 1\}$ and $q_i \in [\![M_i]\!]^k$. If $\rho(q_i) < k$ then Eve has a winning strategy by induction hypothesis. Otherwise, if $\rho(q_i) = k$ we repeat the reasoning.

This strategy is winning for Eve since a play either stays in states of even rank k or switches to a play following a winning strategy for smaller ranks. □

It remains to show that $\mathcal{A}(M) \cap Q_{\leq k} \subseteq [\![M]\!]^k$. For this we will define one logical relation between $\mathcal{D}^k$ and the syntactic model of λY and show a couple of lemmas.

Definition 4.15. We define a logical relation between the model $\mathcal{D}^k$ and closed terms

$$R^k_0 = \{(P, M) : \mathcal{A}(M) \cap Q_{\leq k} \subseteq P\}$$
$$R^k_{A\to B} = \{(f, M) : \forall_{(g,N)\in R_A}.\ (f(g), MN) \in R_B\}\ .$$

Since R^k is a logical relation we have:

Lemma 4.16. *If $M =_{\beta\delta} N$ and $(f, M) \in R^k_A$ then $(f, N) \in R^k_A$.*

The next lemma shows a relation between R^k and R^{k-1}.

Lemma 4.17. *For every type A, $f \in \mathcal{D}^k_A$, $g \in \mathcal{D}^{k-1}_A$:*

- *if $(f, M) \in R^k_A$ then $(f^\downarrow, M) \in R^{k-1}_A$;*
- *if $(g, M) \in R^{k-1}_A$ then $(g^{\uparrow\vee}, M) \in R^k$;*

Proof. The proof is an induction on the size of the type. The base case is when $A = o$.

For the first item suppose $(f, M) \in R^k_A$. By definition, this means $\mathcal{A}(M) \cap Q_{\leq k} \subseteq f$. Then $\mathcal{A}(M) \cap Q_{\leq k-1} \subseteq f \cap Q_{\leq k-1} = f^\downarrow$. So $(f^\downarrow, M) \in R^{k-1}_A$.

For the second item suppose $(g, M) \in R^{k-1}_A$. So $\mathcal{A}(M) \cap Q_{\leq k-1} \subseteq g$. We have $\mathcal{A}(M) \cap Q_{\leq k} \subseteq (\mathcal{A}(M) \cap Q_{\leq k-1}) \cup Q_k \subseteq g \cup Q_k = g^{\uparrow\vee}$.

For the induction step let A be $B \to C$. Let us consider the first item. Suppose $(f, M) \in R^k_{B\to C}$. Take $(h, N) \in R^{k-1}_B$, we need to show that $(f^\downarrow(h), MN) \in R^{k-1}_C$. By the second item of the induction hypothesis we get $(h^{\uparrow\vee}, N) \in R^k_B$. Then $(f(h^{\uparrow\vee}), MN) \in R^k_C$, by the definition of $R^k_{B\to C}$. Using the first item of the induction hypothesis we get $((f(h^{\uparrow\vee}))^\downarrow, MN) \in R^{k-1}_C$. Then using Corollaries 4.2 and 4.3 we obtain $(f(h^{\uparrow\vee}))^\downarrow = f^\downarrow((h^{\uparrow\vee})^\downarrow) = f^\downarrow(h)$.

For the proof of the second item consider $(g, M) \in R^{k-1}_{B\to C}$ and $(h, N) \in R^k_B$. We need to show that $(g^{\uparrow\vee}(h), MN) \in R^k_C$. From the first item of the induction hypothesis we obtain $(h^\downarrow, N) \in R^{k-1}_B$, so $(g(h^\downarrow), MN) \in R^{k-1}_C$. The second item of the induction hypothesis gives us $((g(h^\downarrow))^{\uparrow\vee}, MN) \in R^k_C$. We are done since $g^{\uparrow\vee}(h) = (g(h^\downarrow))^{\uparrow\vee}$ by Corollary 4.3. □

The right to left inclusion of Theorem 4.13 is implied by the following more general statement.

Lemma 4.18. *Let v be a valuation, and let σ be a substitution of closed terms satisfying $(v(x^A), \sigma(x^A)) \in R^k_A$ for every variable x^A in the domain of σ. For every term M of a type A we have $([\![M]\!]^k_v, M.\sigma) \in R^k_A$.*

Proof. The proof is by induction on the structure of M.

If M is a variable then the proof is immediate.

If M is a constant a then we show that $([\![a]\!], a) \in R^k_{o\to o\to o}$. For this we take arbitrary $(S_0, N_0), (S_1, N_1) \in R^k_0$, and we show that $([\![a]\!]^k(S_0, S_1), aN_0N_1) \in R^k_0$. Take $q \in \mathcal{A}(aN_0N_1) \cap Q_{\leq k}$. Let us look at Eve's winning strategy in the acceptance game from q on $BT(aN_0N_1)$. In the first round of this game she chooses some $(T_0, T_1) \in \delta(a, q)$. So $q \in [\![a]\!](T_0, T_1)$. Since her strategy is winning we have $T_0 \subseteq \mathcal{A}(N_0)$ and $T_1 \subseteq \mathcal{A}(N_1)$, and by weakness of the automaton $T_0, T_1 \subseteq Q_{\leq k}$. From the definition of R^k_0 we get $\mathcal{A}(N_0) \cap Q_{\leq k} \subseteq S_0$ and $\mathcal{A}(N_1) \cap Q_{\leq k} \subseteq S_1$. By monotonicity we get the desired $q \in [\![a]\!](S_0, S_1)$.

If M is an application NP then the conclusion is immediate from the definition of R^k_A and the induction hypothesis.

If M is an abstraction $\lambda x.\ N : B \to C$, then we take $(g, P) \in R^k_B$. By induction hypothesis $([\![N]\!]^k_{v[g/x]}, N.\sigma[P/x]) \in R^k_C$. So $([\![M]\!]^k_v(g), MP) \in R^k_C$ by Lemma 4.16.

If $M = Y^{(A\to A)\to A}$. Take $(f, P) \in R^k_{A\to A}$. By Lemma 4.17 we have $(f^{\downarrow}, P) \in R^{k-1}_{A\to A}$. As by the outermost induction hypothesis

$$(\mathrm{fix}^{k-1}_A, Y^{(A\to A)\to A}) \in R^{k-1}_{(A\to A)\to A}\ ,$$

and we obtain $(\mathrm{fix}^{k-1}_A(f^{\downarrow}), YP) \in R^{k-1}_A$. Once again using Lemma 4.17 we can deduce $((\mathrm{fix}^{k-1}_A(f^{\downarrow}))^{\uparrow\vee}, YP) \in R^k_A$. By the choice of (f, P) we obtain

$$(f((\mathrm{fix}^{k-1}_A(f^{\downarrow}))^{\uparrow\vee}), P(YP)) \in R^k_A\ .$$

Since $YP =_{\beta\delta} P(YP)$, we have $(f^i((\mathrm{fix}^{k-1}_A(f^{\downarrow}))^{\uparrow\vee}), YP) \in R^k_A$ for all $i \geq 0$. Since the sequence of $f^i((\mathrm{fix}^{k-1}_A(f^{\downarrow}))^{\uparrow\vee})$ is decreasing, it reaches the fixpoint $\mathrm{fix}^k_A(f)$ in a finite number of steps and $(\mathrm{fix}^k_A(f), YP)$ is in R_A. As (f, P) is an arbitrary element of $R^k_{A\to A}$, this shows that (fix^k_A, Y) is in $R^k_{(A\to A)\to A}$. □

5. From models to type systems

We are now in a position to show that our type system from Figure 3 can reason about the values of λY-terms in a stratified model, cf. Theorem 5.5 below. Thanks to Theorem 4.13 this means that the type system can talk about the acceptance of the Böhm tree of a term by the automaton. This implies the soundness and completeness of our type system, Theorem 3.2.

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 the sake of the 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 later use we also define co-step functions. For d in $\mathcal{L}_1$ and e in $\mathcal{L}_2$, the *step function* $d \rightharpoonup e$ and the *co-step function* $d \rightharpoondown g$ are defined by:

$$(d \rightharpoonup e)(h) = \begin{cases} e \text{ when } d \leq h \\ \bot \text{ otherwise} \end{cases} \qquad (d \rightharpoondown e)(h) = \begin{cases} e \text{ when } h \leq d \\ \top \text{ otherwise .} \end{cases}$$

To emphasize that we work in $\mathcal{D}^k$ we will sometimes write $d \rightharpoonup^k e$ and $d \rightharpoondown^k e$.

Types introduced on page 7 can be meaningfully interpreted at every level of the model. So $[\![t]\!]^k$ will denote the interpretation of t in $\mathcal{D}^k$ defined as follows.

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

Directly from the definition we have $[\![S_1 \cup S_2]\!]^k = [\![S_1]\!]^k \vee [\![S_2]\!]^k$, and $[\![S \to T]\!]^k = [\![S]\!]^k \rightharpoonup^k [\![T]\!]^k$.

The next lemma summarizes basic facts about the interpretation of types. Recall that the application operation $S(T)$ on types (cf. page 8) means $\{t : (U \to t) \in S \wedge U \sqsubseteq T\}$. The proof of the lemma uses Corollaries 4.3 and 4.2.

Lemma 5.1. *For every type A, if $S \subseteq \mathrm{Types}_A$ and $k \leq m$ we have: $[\![S]\!]^k = [\![S \cap \mathrm{Types}^k_A]\!]^k$, $[\![S \cap \mathrm{Types}^k_A]\!]^{k+1} = ([\![S]\!]^k_A)^{\uparrow\wedge}$ and $[\![S]\!]^k = ([\![S]\!]^{k+1})^{\downarrow}$.*

Proof. Consider the first statement. Since $[\![S]\!]^k = \bigvee\{[\![t]\!]^k : t \in S\}$ it is sufficient to show that $[\![t]\!]^k = \bot^k_A$ for $t \notin \mathrm{Types}^k_A$. We do it by induction on the type A.

Suppose that $t \in \mathrm{types}^l_A$ for $l > k$. For the type o it follows directly from the definition that $[\![t]\!]^k = \emptyset$. For A of the form $B \to C$ we know that t is of the form $T \to s$ with $T \subseteq \mathrm{Types}^l_B$ and $s \in \mathrm{types}^l_C$. By induction assumption $[\![s]\!]^k = \bot^k_C$. We get $[\![T \to s]\!]^k = [\![T]\!]^k \rightharpoonup^k [\![s]\!]^k = [\![T]\!]^k \rightharpoonup^k \bot_C = \bot^k_{B\to C}$.

We give the proof of the second statement, the proof of the third is analogous. We prove the result only for elements of types^k_A as the more general one is a direct consequence of that particular case. The proof is by induction on A. The base case is obvious. For A of the form $B \to C$ we have $[\![T \to s]\!]^{k+1}$ is by definition $[\![T]\!]^{k+1} \rightharpoonup^{k+1} [\![s]\!]^{k+1}$ which by induction hypothesis is $([\![T]\!]^k)^{\uparrow\wedge} \rightharpoonup^{k+1} ([\![s]\!]^k)^{\uparrow\wedge}$. We will be done if we show that for every $f \in \mathcal{D}^k_B$ and $g \in \mathcal{D}^k_C$:

$$f^{\uparrow\wedge} \rightharpoonup^{k+1} g^{\uparrow\wedge} = (f \rightharpoonup^k g)^{\uparrow\wedge} \ .$$

Given e in $\mathcal{D}^{k+1}_B$, by Corollary 4.3, we have $(f \rightharpoonup^k g)^{\uparrow\wedge}(e)$ is equal to $((f \rightharpoonup^k g)(e^{\downarrow}))^{\uparrow\wedge}$, and therefore:

$$(f \rightharpoonup^k g)^{\uparrow\wedge}(e) = \begin{cases} g^{\uparrow\wedge} & \text{if } f \leq e^{\downarrow}, \\ \bot^{k+1}_C & \text{otherwise.} \end{cases}$$

Since $f \leq e^{\downarrow}$ iff $f^{\uparrow\wedge} \leq e$, by Corollary 4.2, this proves the desired equality. □

Actually every element of $\mathcal{D}^k$ is the image of some type via $[\![\cdot]\!]^k$: types are syntactic representations of the model. For this we use $\overline{(\cdot)}$ operation (cf. Definition 4.4).

Lemma 5.2. *For every $k \leq m$, every type A and every d in $\mathcal{D}^k_A$ there is $S \subseteq \mathrm{Types}^k_A$ so that $[\![S]\!]^k = d$, and there is $S' \subseteq \mathrm{types}^k_A$ so that $[\![S']\!]^k = \overline{d}$.*

Proof. We proceed by induction on k.

The case where $k = 0$ has been proved in [34].

For the case $k > 0$, as we have seen with Lemma 4.5, that $f = (f^{\downarrow})^{\uparrow\wedge} \vee \overline{f}$. From the induction hypothesis there is $S_1 \subseteq \mathrm{Types}^{k-1}_A$ such that $[\![S_1]\!]^{k-1} = f^{\downarrow}$. By Lemma 5.1 we get $[\![S_1]\!]^k = (f^{\downarrow})^{\uparrow\wedge}$.

It remains to describe $\overline{f}$ with types from $\mathrm{types}^k_{B\to C}$. Take $d \in \mathcal{D}^k_B$ and recall that $\overline{f}(d) = \overline{f(d)}$. By induction hypothesis we have $S_d \subseteq \mathrm{Types}^k_B$ and $S_{\overline{f(d)}} \subseteq \mathrm{types}^k_C$ such that $[\![S_d]\!]^k = d$ and $[\![S_{\overline{f(d)}}]\!]^k = f(d)$. So the set of types $S_d \to S_{\overline{f(d)}}$ is included in $\mathrm{types}^k_{B\to C}$ and $[\![S_d \to S_{\overline{f(d)}}]\!] = d \rightharpoonup^k \overline{f}(d)$. It remains to take $S_2 = \bigcup\{S_d \to S_{\overline{f(d)}} \mid d \in \mathcal{D}^k_B\}$. We can conclude that $S_2 \subseteq \mathrm{types}^k_A$ and $[\![S_2]\!]^k = \overline{f}$. Therefore $[\![S_1 \cup S_2]\!]^k = f$. □

Not only can type represent every element of the model, but also the subsumption rule exactly represents the partial order of the model.

Lemma 5.3. *For each k, A, S and T in Types^k_A, we have that $[\![S]\!]^k \leq [\![T]\!]^k$ iff $S \sqsubseteq_A T$.*

Proof. This is a consequence of the fact that for each k, $\mathcal{D}_k$ can be embedded in the monotone model generated by $\mathcal{P}(Q_{\leq k})$ and that according to [31], the ordering on intersection types simulate the one in monotone models. □

An immediate consequence is that the application that we defined at the level of type simulates the application in the model.

Lemma 5.4. *For $S \subseteq \mathrm{Types}^k_{A\to B}$ and $T \subseteq \mathrm{Types}^k_A$, we have:*

$$[\![S(T)]\!]^k = [\![S]\!]^k([\![T]\!]^k) \ .$$

Proof. By definition $S(T) = \{t \mid (U \to t) \in S \text{ and } U \sqsubseteq T\}$, but $[\![S]\!]^k = \bigvee\{[\![U \to t]\!]^k \mid U \to t \in S\} = \bigvee\{[\![U]\!]^k \to^k [\![t]\!]^k \mid U \to t \in S\}$ and thus $[\![S]\!]^k([\![T]\!]^k) = \bigvee\{[\![t]\!]^k \mid U \to t \in S \text{ and } [\![U]\!]^k \leq [\![T]\!]^k\}$. From Lemma 5.3, $U \sqsubseteq T$ iff $[\![U]\!]^k \leq [\![T]\!]^k$ and we then obtain the expected idendity: $[\![S]\!]^k([\![T]\!]^k) = \{t \mid (U \to t) \in S \text{ and } U \sqsubseteq T\} = S(T)$. □

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 5.5. *For $k = 0, \ldots, m$ and $S \subseteq \mathrm{Types}^k$:*

$$[\![M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k \quad \textit{iff} \quad \Gamma \vdash M \geq S \textit{ is derivable.}$$

The above theorem implies Theorem 3.2 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 4.13 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 $q \in \mathcal{A}(M)$.

The theorem is proved by the following two lemmas.

Lemma 5.6. *If $\Gamma \vdash M \geq S$ is derivable, then for every $k \leq m$: $[\![M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k$.*

Proof. This proof is done by a simple induction on the structure of the derivation of $\Gamma \vdash M \geq S$. For most of the rules, the conclusion follows immediately from the induction hypothesis (using the Lemmas 5.1, 5.3 and 5.4). We shall only treat here the case of the rules *Y odd* and *Y even*.

In the case of *Y odd*, when we derive $\Gamma \vdash Yx.M \geq S(T)$ from $\Gamma \vdash \lambda x.M \geq S$ and $\Gamma \vdash Yx.M \geq T$, the induction hypothesis gives that for every k, $[\![\lambda x.M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k$ and $[\![Yx.M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![T]\!]^k$. Therefore $[\![Yx.M]\!]^k_{[\![\Gamma]\!]^k} = [\![(\lambda x.M)(Yx.M)]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k([\![T]\!]^k) = [\![S(T)]\!]^k$, using Lemma 5.4.

In the case of *Y even* we consider the case $k = 2l$. Let ν_{k-1} stand for $[\![\Gamma]\!]^{k-1}$ and ν_k for $[\![\Gamma]\!]^k$.

By induction hypothesis we have $[\![Yx.M]\!]^{k-1}_{\nu_{k-1}} \geq [\![T]\!]^{k-1}$. Since Lemma 4.10 implies $([\![Yx.M]\!]^{k-1}_{\nu_{k-1}}, [\![Yx.M]\!]^k_{\nu_k}) \in \mathcal{L}^k$, we have $[\![Yx.M]\!]^k_{\nu_k} \geq ([\![T]\!]^{k-1})^{\uparrow\wedge}$ by Lemma 4.1. By Lemma 5.1 we know $([\![T]\!]^{k-1})^{\uparrow\wedge} = [\![T]\!]^k$. In consequence we have

$$[\![\lambda x.M]\!]^k_{\nu_k}([\![T]\!]^k) \geq [\![T]\!]^k.$$

Also by induction hypothesis we have $[\![\lambda x.M]\!]^k_{\nu_k} \geq [\![(S \cup T) \to S]\!]^k$. This means that $[\![\lambda x.M]\!]^k_{\nu_k}([\![S \cup T]\!]^k) \geq [\![S]\!]^k$. Put together with what we have concluded about $[\![T]\!]^k$ we get

$$[\![\lambda x.M]\!]^k_{\nu_k}([\![S \cup T]\!]^k) \geq [\![S \cup T]\!]^k.$$

Now we use Lemma 4.9 which tells us that

$$[\![Yx.M]\!]^k_{\nu_k} = \bigvee \{d \mid [\![\lambda x.M]\!]^k(d) \geq d \text{ and } d^{\downarrow} = [\![Yx.M]\!]^{k-1}_{\nu_{k-1}}\} .$$

This gives us immediately the desired $[\![Yx.M]\!]^k_{\nu_k} \geq [\![S \cup T]\!]^k$. □

Lemma 5.7. *Given a type $S \subseteq \text{Types}^k$, if $[\![M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S]\!]^k$ then $\Gamma \vdash M \geq S$.*

Proof. This theorem is proved by induction on the pairs (M, k) ordered component-wise. Suppose that the statement is true for M, we are going to show that it is true for $Yx.M$, the other cases are straightforward.

The first observation is that, if $T \to S$ is such that $[\![\lambda x.M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![T \to S]\!]^k$, then $\Gamma \vdash \lambda x.M \geq T \to S$ is derivable. Indeed, since, letting $\nu = [\![\Gamma, x \geq T]\!]^k$, if $[\![M]\!]^k_{\nu} \geq [\![S]\!]^k$ holds then, $\Gamma, x \geq T \vdash M \geq S$ is derivable by induction hypothesis. So $\Gamma \vdash \lambda x.M \geq T \to S$ is derivable.

There are now two cases depending on the parity of k. First let us assume that k is even. Suppose $[\![Yx.M]\!]^k_{[\![\Gamma]\!]^k} = [\![S \cup T]\!]^k$ where $S \subseteq \text{types}^k$ and $T \subseteq \text{Types}^{k-1}$. Lemma 5.2 guaranties the existence of such S and T as every element of $\mathcal{D}^k$ is expressible by a set of types. We have $[\![\lambda x.M]\!]^k_{[\![\Gamma]\!]^k} \geq [\![S \cup T]\!]^k \to^k [\![S \cup T]\!]^k$. By the above we get that $\Gamma \vdash \lambda x.M \geq (S \cup T) \to (S \cup T)$ is derivable and thus $\Gamma \vdash \lambda x.M \geq (S \cup T) \to S$ is derivable as well. We also have $[\![Yx.M]\!]^{k-1} \geq [\![T]\!]^{k-1}$ which gives that $\Gamma \vdash Yx.M \geq T$ is derivable. This allows us to derive $\Gamma \vdash Yx.M \geq S \cup T$. Using the subsumption rule and the fact that the subsumption reflects the order in the model (Lemma 5.3), every other judgment $\Gamma \vdash Yx.M \geq U$ where $[\![U]\!]^k \leq [\![Yx.M]\!]^k_{[\![\Gamma]\!]^k}$ is derivable.

Now consider the case where k is odd. Suppose $[\![Yx.M]\!]^k_{[\![\Gamma]\!]^k} = [\![S \cup T]\!]^k$ with $T \subseteq \text{Types}^{k-1}$ and $S \subseteq \text{types}^k$. By induction hypothesis on k, we have that $\Gamma \vdash M \geq T$ is derivable. Take $d = [\![\lambda x.M]\!]^k_{[\![\Gamma]\!]_k}$. Lemma 5.2 guarantees us a set of types $U \subseteq \text{Types}^k$ such that $[\![U]\!]^k = d$. By the observation we have made above, there is a derivation of $\Gamma \vdash \lambda x.M \geq U$. Then iteratively using the rule Y *odd* we compute the least fixpoint by letting $U^0(T) = T$ and $U^{n+1}(T) = U(U^n(T))$. As in the case above, we can now conclude that for every $V \subseteq \text{Types}^k$, if $[\![V]\!]^k \leq [\![Yx.M]\!]^k_{[\![\Gamma]\!]^k}$, then $\Gamma \vdash Yx.M \geq U$ is derivable. □

As we have seen, the applicative structure $\mathcal{D}^k_A$ is a lattice, therefore each construction can be dualized: in Abramsky's methodology, this consists in considering $\wedge$-prime elements of the models, meets and co-step functions instead of $\vee$-primes, joins and step functions. It is worth noticing that dualizing at the level of the model amounts to dualizing the automaton. So, in particular, we can define a system so that $BT(M)$ is not accepted by $\mathcal{A}$ from state q iff $\Gamma \vdash M \not\geq q$ is derivable. While the first typing system establishes positive facts about the semantics, the second one refutes them. For this, we use the same syntax to denote types, but we give types a different semantics that is dual to the first semantics we have used. The

notation, $\geq$ and $\not\geq$, we have used for the two type systems is motivated by this duality.

$$\langle\!\langle q \rangle\!\rangle^k = Q_{\leq k} - \{q\} \ ,$$
$$\langle\!\langle S \to f \rangle\!\rangle^k = \Big(\bigwedge \{ \langle\!\langle g \rangle\!\rangle^k : g \in S \}\Big) \rightharpoonup^k \langle\!\langle f \rangle\!\rangle^k \ .$$

The dual type system is presented in Figure 4. The notation is as before but we use $\not\geq$ instead of $\geq$. Similarly to the definition of $[\![\cdot]\!]^k$, we write $\langle\!\langle S \rangle\!\rangle^k$ for $\bigwedge\{\langle\!\langle s \rangle\!\rangle^k : s \in S\}$ and we have that $\langle\!\langle T \to S \rangle\!\rangle^k = \langle\!\langle T \rangle\!\rangle^k \rightharpoonup^k \langle\!\langle S \rangle\!\rangle^k$. We also need to redefine $S(T)$ to be $\{s : U \to s \in S \wedge U \sqsupseteq T\}$. By duality, from Theorem 5.5 we obtain:

Theorem 5.8. *For $S \subseteq \mathrm{Types}^k$: $\Gamma \vdash M \not\geq S$ is derivable iff $[\![M]\!]^k_{[\![\Gamma]\!]^k} \leq \langle\!\langle S \rangle\!\rangle^k$.*

Together Theorems 5.5 and 5.8 give a characterization by typing of $[\![M]\!] = L(\mathcal{A})$, that is the set of states from which our fixed automaton $\mathcal{A}$ accepts $BT(M)$.

Corollary 5.9. *For a closed term M of type o:*

$$[\![M]\!] = [\![S]\!] \quad \textit{iff} \quad \textit{both} \quad \vdash M \geq S \ \textit{and} \ \ \vdash M \not\geq (Q - S).$$

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 [34] 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 wMSOL 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. Kobayashi and Ong [21] were the first to approach higher-order verification of MSOL properies through typing. In their type system derivations are graphs, or infinite trees, and their validity is defined via some regular acceptance condition on infinite paths. Their type system handles only closed terms of type o, and fixpoint are handled via the condition on infinite paths. Tsukada and Ong have recently proposed a higher-order analogue of this system [38]. The typability is defined in a standard way as the existence of a finite derivation. The semantics of the fixpoint combinator is defined via some special games. The soundness and completeness proofs use a syntactic approach. In our case, thanks to the restriction to wMSO, we can use standard fixpoint rules to handle the fixpoint combinator, we also obtain a model allowing us to prove soundness and completeness using quite standard techniques.

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 [23]. 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 [33] theorems for wMSOL. 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 MSOL 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 [34].

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 o over the fixed vocabulary: M satisfies $\widehat{\varphi}$ iff the Böhm tree of M satisfies φ. Since the MSOL theory of a term, that is a finite graph, is decidable, the transfer theorem implies decidability of MSOL theory of Böhm trees of λY-terms. As shown in [33] it gives also a number of other results.

A transfer theorem for wMSOL can be deduced from our model construction. For every wMSOL 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 wMSOL formula α_d such that for every term M of type o 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 [34] 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] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991.

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

[3] R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[4] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symb. Log.*, 4:931–940, 1983.

[5] S. L. Bloom and Z. Ésik. *Iteration Theories: The Equational Logic of Iterative Processes*. EATCS Monographs in Theoretical Computer Science. Springer, 1993.

[6] Stephen L. Bloom and Zoltàn Ésik. Fixed-point operations on CCC's. part I. *Theoretical Computer Science*, 155:1–38, 1996.

[7] A. Blumensath. An algebraic proof of Rabin's tree theorem. *Theor. Comput. Sci.*, 478:1–21, 2013.

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

[9] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24. ACM, 2013.

[10] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl, 2013.

[11] W. Chen and M. Hofmann. Buchi abstraction. In *LICS*, pages 51:1–51:10, 2014.

[12] R. Grabowski, M. Hofmann, and K. Li. Type-based enforcement of secure programming guidelines - code injection prevention at SAP. In *Formal Aspects in Security and Trust*, volume 7140 of *LNCS*, pages 182–197, 2011.
[13] A. Haddad. Model checking and functional program transformations. In *FSTTCS*, volume 24 of *LIPIcs*, pages 115–126, 2013.
[14] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.
[15] Gerd G. Hillebrand. *Finite Model Theory in the Simply Typed Lambda Calculus*. PhD thesis, Department of Computer Science, Brown University, Providence, Rhode Island 02912, 1994.
[16] J. R. Hindley and J. P. Seldin. *Lambda-Calculus and Combinators*. Cambridge University Press, 2008.
[17] A. S. A. Jeffrey. LTL types FRP: Linear-time Temporal Logic propositions as types, proofs as functional reactive programs. In *ACM Workshop Programming Languages meets Program Verification*, 2012.
[18] A. S. A. Jeffrey. Functional reactive types. In *LICS*, pages 54:1–54:9, 2014.
[19] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428, 2009.
[20] N. Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20–89, 2013.
[21] N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.
[22] N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *POPL*, pages 495–508, 2010.
[23] Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP II 2009*, volume 5556 of *LNCS*, pages 223–234. Springer, 2009.
[24] R. Loader. Finitary PCF is not decidable. *Theor. Comput. Sci.*, 266(1-2):341–364, 2001.
[25] M. Naik and J. Palsberg. A type system equivalent to a model checker. *ACM Trans. Program. Lang. Syst.*, 30(5), 2008.
[26] F. Nielson and H. R. Nielson. Type and effect systems. In *Correct System Design: Recent Insight and Advances*, volume 1710 of *LNCS*, pages 114–136. Springer-Verlag, 1999.
[27] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.
[28] C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *POPL*, pages 587–598, 2011.
[29] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.
[30] S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, pages 61–72. ACM, 2014.
[31] S. Salvati, G. Manzonetto, M. Gehrke, and H. Barendregt. Loader and Urzyczyn are logically related. In *ICALP*, volume 7392 of *LNCS*, pages 364–376. Springer, 2012.
[32] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP*, volume 6756 of *LNCS*, pages 162–173, 2011.
[33] S. Salvati and I. Walukiewicz. Evaluation is MSOL-compatible. In *FSTTCS*, volume 24 of *LIPIcs*, pages 103–114, 2013.
[34] S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, volume 7941 of *LNCS*, pages 189–204, 2013.
[35] R. Statman. Completeness, invariance and lambda-definability. *J. Symb. Log.*, 47(1):17–26, 1982.
[36] K. Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl, 2012.
[37] Y. Tobita, T. Tsukada, and N. Kobayashi. Exact flow analysis by higher-order model checking. In *FLOPS*, volume 7294 of *LNCS*, pages 275–289, 2012.
[38] T. Tsukada and C.-H. L. Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *LICS*, pages 78:1–78:10, 2014.