Intersection Types and Counting*

Paweł Parys
University of Warsaw, Poland
parys@mimuw.edu.pl

We present a new approach to the following meta-problem: given a quantitative property of trees, design a type system such that the desired property for the tree generated by an infinitary ground λ-term corresponds to some property of a derivation of a type for this λ-term, in this type system.

Our approach is presented in the particular case of the language finiteness problem for nondeterministic higher-order recursion schemes (HORSes): given a nondeterministic HORS, decide whether the set of all finite trees generated by this HORS is finite. We give a type system such that the HORS can generate a tree of an arbitrarily large finite size if and only if in the type system we can obtain derivations that are arbitrarily large, in an appropriate sense; the latter condition can be easily decided.

1 Introduction

In this paper we consider simply-typed λY-calculus, that is an extension of the simply-typed λ-calculus by a fixpoint operator Y. A term P of the simply typed λY-calculus that is of sort[1] o can be used to generate an infinite tree $BT(P)$, called the Böhm tree of P. Trees generated by terms of simply-typed λY-calculus can be used to faithfully represent the workflow of programs in languages with higher-order functions. Traditionally, Higher Order Recursive Schemes (HORSes) are used for this purpose [8, 11, 16, 15]; this formalism is equivalent to simply-typed λY-calculus, and the translation between them is rather straightforward [20]. Collapsible Pushdown Systems [9] and Ordered Tree-Pushdown Systems [7] are other equivalent formalisms.

Intersection type systems were intensively used in the context of HORSes, for several purposes like model-checking [12, 15, 5, 19], pumping [13], transformations of HORSes [14, 6], etc. Interestingly, constructions very similar to intersection types were used also on the side of collapsible pushdown systems; that were alternating stack automata [4], and types of stacks [17, 10].

In this paper we show how intersection types can be used for deciding quantitative properties of trees generated by λY-terms. We concentrate on the language finiteness problem for nondeterministic HORSes: given a nondeterministic HORS, decide whether the set of all finite trees generated by this HORS is finite. This can be restated in the world of λY-terms (or standard, deterministic HORSes), generating a single infinite tree in which a new br ("branch") symbol of rank 2 may be present. The br symbol denotes the nondeterministic choice, so to obtain a finite tree generated by the original HORS we just need to recursively choose in every br-labeled node which of the two subtrees we want to consider. Then the language finiteness problem asks whether the set of all finite trees obtained this way is finite.

The difficulty of this problem lies in the fact that sometimes the same finite tree may be found in infinitely many different places of $BT(P)$ (i.e. generated by a nondeterministic HORS in many ways); thus the actual property to decide is whether there is a common bound on the size of every of these trees.

*Work supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

[1] We use the word "sort" instead of the usual "type" to avoid confusion with intersection types introduced in this paper.

Submitted to:
ITRS 2016

This makes the problem inaccessible for standard methods used for analyzing HORSes, as they usually concern only regular properties of the Böhm tree, while boundedness is a problem of different kind. The same difficulty was observed in [13], where they prove a pumping lemma for deterministic HORSes, while admitting (Remark 2.2) that their method is too weak to reason about nondeterministic HORSes.

In order to solve the language finiteness problem, we present appropriate intersection type system, where derivations are annotated by flags and markers of multiple kinds. The key property of this type system is that the number of flags in a type derivation for a λY-term P approximates the size of some finite tree obtained by resolving nondeterministic choices in the infinite tree $BT(P)$. In consequence, there are type derivations using arbitrarily many flags if, and only if, the answer to the language finiteness problem is "no".

The language finiteness problem was first attacked in [1] (for safe HORSes only), but then their algorithm turned out to be incorrect [2]. Up to our knowledge, the only known solution of this problem follows from a recent decidability result for the diagonal problem [6]. This problem asks, given a nondeterministic HORS and a set of letters Σ, whether for every $n \in \mathbb{N}$ the HORS generates a finite tree in which every letter from Σ appears at least n times. Clearly, a nondeterministic HORS generates arbitrarily large trees exactly when for some letter a it generates trees having arbitrarily many a letters, i.e. when the answer to the diagonal problem for $\Sigma = \{a\}$ is "yes".

Our type system is, to some extent, motivated by the algorithm of [6] solving the diagonal problem. This algorithm works by repeating two kinds of transformations of HORSes. The first of them turns the HORS into a HORS generating trees having only a fixed number of branches, one per each letter from Σ (i.e. one branch in our case of $|\Sigma| = 1$). The branches are chosen nondeterministically out of some tree generated by the original HORS; for every $a \in \Sigma$ there is a choice witnessing that a appeared many times in the original tree. Then such a HORS of the special form is turned into a HORS that is of order lower by one, and generates trees having the same nodes as trees generated by the original HORS, but arranged differently (in particular, the new trees may have again arbitrarily many branches). After finitely many repetitions of this procedure, a HORS of order 0 is obtained, and the diagonal problem becomes easily decidable. In some sense we want to do the same, but instead of applying all these transformations one by one, we simulate all of them simultaneously in a single type derivation. In this derivation, for each order n, we allow to place arbitrarily one marker "of order n", what corresponds to the nondeterministic choice of one branch in the n-th step of the previous algorithm. We also place some flags "of order n", in places that correspond to nodes remaining after the n-th step of the previous algorithm.

The idea of using intersection types for counting is not completely new. In [18] there is a type system that, essentially, allows to estimate the size of the β-normal form of a λ-term just by looking at (the number of some flags in) a derivation of a type for this term. A similar idea, but for higher-order pushdown automata, is present in [17], where we can estimate the number $\sharp$ symbols appearing on a particular, deterministically chosen branch of the generated tree. This previous approach also uses intersection types, where the derivations are marked with just one kind of flags, denoting "productive" places of a term (oppositely to our approach, where we have different flags for different orders, and we also have markers). The trouble with the "one-flag" approach is that it works well only in a completely deterministic setting, where looking independently at each node of the Böhm tree we know how it contributes to the result; the method stops working (or at least we do not know how to prove that it works) in our situation, where we first nondeterministically perform some guesses in the Böhm tree, and only after that we want to count something that depends on the chosen values.

Acknowledgements. I would like to thank Szymon Toruńczyk for stimulating discussions.

2 Preliminaries

Trees. Let Σ be a *ranked alphabet*, i.e., a set of symbols together with a rank function assigning a nonnegative integer to each of the symbols. We assume that Σ contains a distinguished symbol br of rank 2, used to denote nondeterministic choices. A Σ-*labeled* tree is a tree that is rooted (there is a distinguished root node), node-labeled (every node has a label from Σ), ranked (a node with label of rank n has exactly n children), and ordered (children of a node of rank n are numbered from 1 to n).

When t is a Σ-labeled tree t, by $\mathscr{L}(t)$ we denote the set of all finite trees that can be obtaining by choosing in every br-labeled node of t which of the two subtrees we want to consider. More formally, we consider the following nondeterministic process transforming trees, starting from t: take any br-labeled node x of the current tree, choose some of its children y, and replace the subtree starting in x by the subtree starting in y (removing x and the other subtree of x). Then, $\mathscr{L}(t)$ contains all trees that are finite, do not use the br label, and can be obtained from t by the above process after finitely many steps.

Infinitary λ-calculus. The set of *sorts* (a/k/a simple types), constructed from a unique basic sort o using a binary operation $\to$, is defined as usual. The order of a sort is defined by: $ord(o) = 0$, and $ord(\alpha\to\beta) = \max(1+ord(\alpha), ord(\beta))$.

We consider infinitary, sorted λ-calculus. *Infinitary λ-terms* (or just *λ-terms*) are defined by coinduction, according to the following rules:

- if $a \in \Sigma$ is a symbol of rank r, and $P_1^o, \ldots, P_r^o$ are λ-terms, then $(a\,P_1^o \ldots P_r^o)^o$ is a λ-term,
- for every sort α there are infinitely many variables $x^\alpha, y^\alpha, z^\alpha, \ldots$; each of them is a λ-term,
- if $P^{\alpha\to\beta}$ and Q^α are λ-terms, then $(P^{\alpha\to\beta}\,Q^\alpha)^\beta$ is a λ-term, and
- if P^β is a λ-term and x^α is a variable, then $(\lambda x^\alpha.P^\beta)^{\alpha\to\beta}$ is a λ-term.

We naturally identify α-equivalent λ-terms. We often omit the sort annotations of λ-terms, but we keep in mind that every λ-term (and every variable) has a particular sort. A λ-term P is *closed* if it has no free variables. Notice that, for technical convenience, a symbol of positive rank is not a term itself, but always comes with arguments. This is not a restriction, since e.g. instead of a unary symbol a one may use the term $\lambda x.a\,x$.

The *complexity* of a λ-term P is the smallest number m such that the order of all subterms of P is at most m. We restrict ourselves to λ-terms that have finite complexity.

Böhm Trees. We consider Böhm trees only for closed λ-terms of sort o. For such a term P, its *Böhm tree* $BT(P)$ is constructed by coinduction, as follows: if there is a sequence of β-reductions from P to a λ-term of the form $a\,P_1 \ldots P_r$ (where a is a symbol), then the root of the tree t has label a and r children, and the subtree starting in the i-th child is $BT(P_i)$. If there is no sequence of β-reductions from P to a λ-term of the above form, then $BT(P)$ is the full binary tree with all nodes labeled by br.[2] By $\mathscr{L}(P)$ we denote $\mathscr{L}(BT(P))$.

λY-calculus. The syntax of λY-calculus is the same as of finite λ-calculus, extended by symbols $Y^{(\alpha\to\alpha)\to\alpha}$, for each sort α. A term of λY-calculus is seen as a term of infinitary λ-calculus if we replace each symbol $Y^{(\alpha\to\alpha)\to\alpha}$ by the unique term Y such that Y is equal to $\lambda x^{\alpha\to\alpha}.x(Y\,x)$. In this way, we view λY-calculus as a fragment of infinitary λ-calculus.

[2] Usually one uses a special label $\bot$ of rank 0 for this purpose, but from the perspective of our problem both definitions are equivalent.

It is standard to convert a nondeterministic HORS $\mathscr{G}$ into a closed λY-term P^o such that $\mathscr{L}(P)$ is exactly the set of all finite trees generated by $\mathscr{G}$. The following theorem, which is our main result, states that the *language finiteness problem* is decidable.

Theorem 1. *Given a closed λY-term P of sort o, one can decide whether $\mathscr{L}(P)$ is finite.*

3 Intersection Type System

In this section we introduce a type system that allows to determine the desired property: whether in $\mathscr{L}(P)$ there is an arbitrarily large tree.

Type Judgments. For every sort α we define the set $\mathscr{T}^\alpha$ of *types* of sort α, and the set $\mathscr{F}^\alpha$ of *full types* of sort α. This is done as follows, where $\mathscr{P}$ denotes the powerset:

$$\mathscr{T}^{\alpha\to\beta} = \mathscr{P}(\mathscr{F}^\alpha_{ord(\alpha\to\beta)}) \times \mathscr{T}^\beta\,, \qquad \mathscr{T}^o = o\,,$$
$$\mathscr{F}^\alpha_k = \{k\} \times \mathscr{P}(\{0,\ldots,k-1\}) \times \mathscr{P}(\{0,\ldots,k-1\}) \times \mathscr{T}^\alpha\,, \qquad \mathscr{F}^\alpha = \bigcup_{k\in\mathbb{N}} \mathscr{F}^\alpha_k\,.$$

A type $(T,\tau) \in \mathscr{T}^{\alpha\to\beta}$ is denoted as $T\to\tau$. A full type $\hat\tau = (k,F,M,\tau) \in \mathscr{F}^\alpha_k$ consists of its order k, a set F of flag orders, a set M of marker orders, and a type τ; we write $ord(\hat\tau) = k$. In order to distinguish types from full types, the latter are denoted by letters with a hat, like $\hat\tau$.

A *type judgment* is of the form $\Gamma \vdash P : \hat\tau \triangleright c$, where Γ, called a *type environment*, is a function that maps every variable x^α to a subset of $\mathscr{F}^\alpha$, P is a term, $\hat\tau$ is a full type of the same sort as P (i.e. $\hat\tau \in \mathscr{F}^\beta$ when P is of sort β), and $c \in \mathbb{N}$.

Intuitions. As usual for intersection types, the intuitive meaning of a type $T\to\tau$ is that a term having this type can return a term having type τ, while taking an argument for which we can derive all full types from T. Moreover, in $\mathscr{T}^o$ there is just one type o, that can be assigned to every term of sort o. Thus the only novelty of the type system is in using flags and markers: we derive full types (m,F,M,τ), where the sets F and M store some information about flags and markers.

Let us first explain the behavior of markers and of the set M. Markers (and flags as well) are indexed by a number, called order. For each $n \in \{0,\ldots,m-1\}$ there is at most one marker of order n. We may put these markers in (almost) arbitrary leaves of a derivation. While deriving a full type (m,F,M,τ), in the set M we store the orders of markers used in the derivation, together with those provided by free variables (i.e. we imagine that some derivations, specified by the type environment, are already substituted in our derivation for free variables). The set M, however, does not include markers provided by arguments of the term.

Flags are handled in a slightly different way than markers. For every $n \in \{0,\ldots,m\}$ arbitrarily many flags of order n may be placed in a derivation. The set F (unlike M) carries the information not only about flags placed in the derivation itself and those provided by free variables, but also about those provided by arguments. We, however, do not want to propagate the information about flags of order n when the marker of order n is visible (when $n \in M$), and thus in such situation we remove n from the set F. While markers could be distributed arbitrarily, flags are placed in a very particular way. Recall that while deriving a type we aim to describe some finite tree obtained by resolving nondeterministic choices in some way. Flags of order 0 are put in all those places of the derivation that correspond to symbols used for generating those nodes of the Böhm tree that are taken to this finite tree. For $n \geq 1$, we place a

flag of order n in all those places where the information about a flag of order $n-1$ reaches a place from which a marker of order $n-1$ is visible (i.e. where $n-1 \in F \cap M$).

Notice that, while deriving a full type (m,F,M,τ), in the sets F and M we can store only numbers smaller than m. Possible marker orders are indeed $0,\dots,m-1$, but possible flag orders are $0,\dots,m$. In the set F we keep information only about flags of order smaller than m. Simultaneously, the number of order-m flags present in the derivation is computed precisely, and stored in the number c, called the *flag counter*.

We have to mention that flags and markers of different orders are propagated differently during an application. Namely, when PQ is an application, and some n belongs to F or M in a derivation for Q, then this n is provided to P only when $n < ord(P)$; flags and markers of higher orders placed in an argument of P are not visible for P.

Type System. Before giving rules of the type system, we need few definitions. We use the symbol $\uplus$ to denote disjoint union. When $A \subseteq \mathbb{N}$ and $n \in \mathbb{N}$, we write $A\restriction_{<n}$ for $\{k \in A \mid k < n\}$, and similarly $A\restriction_{\geq n}$ for $\{k \in A \mid k \geq n\}$. We use ε to denote the type environment mapping every variable to $\emptyset$.

Let us now say how a type environment Γ from the conclusion of a rule may be split into type environments $(\Gamma_i)_{i\in I}$ used in premisses of the rule: we say that $Split(\Gamma \mid (\Gamma_i)_{i\in I})$ holds if and only if for every variable x it holds $\Gamma_i(x) \subseteq \Gamma(x)$ for every $i \in I$, and every full type from $\Gamma(x)$ providing some markers (i.e. (k,F,M,τ) with $M \neq \emptyset$) appears in some $\Gamma_i(x)$. Full types with empty M may be discarded and duplicated freely. This definition forbids to discard full types with nonempty M, and from elsewhere it will follow that they cannot be duplicated. Notice that $Split(\Gamma \mid \Gamma')$ describes how a type environment can be weakened.

We also need to say how flags and markers from premisses contribute to the conclusion. We thus say that $Comp_m(M,F,c \mid ((F_i,c_i))_{i\in I})$ holds when

$$F = \{n \in \{0,\dots,m-1\} \mid f_n > 0 \wedge n \notin M\}, \qquad c = f'_m + \sum_{i\in I} c_i, \qquad \text{where, for } n \in \{0,\dots,m\},$$

$$f_n = f'_n + \sum_{i\in I} |F_i \cap \{n\}|, \qquad f'_n = \begin{cases} f_{n-1} & \text{if } n-1 \in M, \\ 0 & \text{otherwise.} \end{cases}$$

Here, the number f'_n contains the number of order-n flags put in the current place, while f_n additionally counts the number of premisses in which such a flag is visible.

We are ready to describe rules of the type system. They correspond to particular constructs of λ-calculus:

$$\frac{\begin{array}{c}\Gamma_i \vdash P_i : (m,F_i,M_i,o) \triangleright c_i \text{ for each } i \in \{1,\dots,r\} \qquad M = M' \uplus M_1 \uplus \dots \uplus M_r \\ (m=0) \Rightarrow (F' = \emptyset \wedge c' = 1) \qquad (m>0) \Rightarrow (F' = \{0\} \wedge c' = 0) \qquad (r>0) \Rightarrow (M' = \emptyset) \\ a \neq \mathsf{br} \qquad Split(\Gamma \mid \Gamma_1,\dots,\Gamma_r) \qquad Comp_m(M,F,c \mid (F',c'),(F_1,c_1),\dots,(F_r,c_r))\end{array}}{\Gamma \vdash a P_1 \dots P_r : (m,F,M,o) \triangleright c} \text{ (Con)}$$

$$\frac{\Gamma \vdash P_i : \hat{\tau} \triangleright c \qquad i \in \{1,2\}}{\Gamma \vdash \mathsf{br} P_1 P_2 : \hat{\tau} \triangleright c} \text{ (Br)} \qquad \frac{Split(\Gamma \mid \varepsilon[x \mapsto \{(k,F,M',\tau)\}]) \qquad M\restriction_{<k} = M'}{\Gamma \vdash x : (m,F,M,\tau) \triangleright 0} \text{ (Var)}$$

$$\frac{\Gamma'[x \mapsto T] \vdash P : (m,F,M,\tau) \triangleright c \qquad Split(\Gamma \mid \Gamma') \qquad \Gamma'(x) = \emptyset}{\Gamma \vdash \lambda x.P : (m,F,M \setminus \bigcup_{(k,F',M',\sigma)\in T} M', T \to \tau) \triangleright c} (\lambda)$$

$$\frac{\begin{array}{c}\Gamma' \vdash P : (m, F', M', \{(ord(P), F_i\restriction_{<ord(P)}, M_i\restriction_{<ord(P)}, \tau_i) \mid i \in I\}) \to \tau) \triangleright c' \\ \Gamma_i \vdash Q : (m, F_i, M_i, \tau_i) \triangleright c_i \text{ for each } i \in I \qquad M = M' \uplus \biguplus_{i\in I} M_i \\ ord(P) \leq m \qquad Split(\Gamma \mid \Gamma', (\Gamma_i)_{i\in I}) \qquad Comp_m(M, F, c \mid (F', c'), ((F_i\restriction_{\geq ord(P)}, c_i))_{i\in I})\end{array}}{\Gamma \vdash PQ : (m, F, M, \tau) \triangleright c} \ (@)$$

All type derivations are assumed to be finite. Since we intend to store in the set M markers contained in the derivation itself and those provided by free variables, but not those provided by arguments, in the (λ) rule we remove markers provided by x. This operation makes sense only because there is at most one marker of every order, so markers provided by x cannot be provided by any other free variable nor placed in the derivation itself. Let us emphasize that in the (CON) rule we have a disjoint union $M_1 \uplus \cdots \uplus M_r$, and similarly $M' \uplus \biguplus_{i\in I} M_i$ in the (@) rule, what ensures that a marker of any order may come only from one of the premisses. Notice that the order m of derived full types is the same in the whole derivation. On the other hand, the order k of full types assigned by a type environment to a variable x equals to the order of the subterm $\lambda x.P$ that bounds this variable (and hence is always greater than $ord(x)$). The (VAR) rule allows to have in the resulting set M some numbers that do not come from the set M' assigned to x by the type environment; these are the orders of markers placed in the leaf using this rule. Notice, however, that we allow here only orders not smaller than k (which is the order of $\lambda x.P$). Markers of arbitrary order may be placed (via the set M') while using the (CON) rule for a symbol of rank 0.

The key property of the type system is described by the following theorem.

Theorem 2. *Let P be a closed term of sort o and complexity m. Then $\mathscr{L}(P)$ is infinite if and only if for arbitrarily large c we can derive $\varepsilon \vdash P : (m, \emptyset, \{0, \ldots, m-1\}, o) \triangleright c$.*

The left-to-right implication of Theorem 2 (completeness of the type system) is shown in Section 4, while the opposite implication (soundness of the type system) in Section 5.

Effectiveness. Denote $\hat{\rho}_m = (m, \emptyset, \{0, \ldots, m-1\}, o)$. Let us now see how Theorem 1 follows from Theorem 2, i.e. how given a term λY-term P we can check whether $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ can be derived for arbitrarily large c. We say that two type judgements are equivalent if they differ only in the value of the flag counter. Let us consider all derivations of $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ in which on every branch there are at most two type judgements from every equivalence class. There are only finitely many such derivations (and thus we can compute all of them), since they use only type judgements $\Gamma \vdash Q : \hat{\tau} \triangleright c$ with Q being a subterm of P and with $\Gamma(x) \neq \emptyset$ only for variables x appearing in P, while a finite λY-term, even when seen as an infinitary λ-term, has only finitely many subterms. Then, a simple pumping argument shows that $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ can be derived for arbitrarily large c if and only if there is a derivation of the above form in which on some branch there are two equivalent type judgements with different values of the flag counter.

4 Completeness

The proof of the left-to-right implication of Theorem 2 is divided into the following three lemmata. We say that a β-reduction $P \to_\beta Q$ *is of order n* if it concerns a redex $(\lambda x.R)\,S$ such that $ord(\lambda x.R) = n$. In this situation the order of x is at most $n-1$, but may be smaller (when other arguments of R are of order $n-1$).

Lemma 3. *Let P be a closed term of sort o and complexity m, and let $t \in \mathscr{L}(P)$. Then there exist terms $Q_m, Q_{m-1}, \ldots, Q_0$ such that $P = Q_m$, and for every $k \in \{1, \ldots, m\}$ the term Q_{k-1} can be reached from Q_k using only β-reductions of order k, and we can derive $\varepsilon \vdash Q_0 : \hat{\rho}_0 \triangleright |t|$.*

Lemma 4. *Suppose that we can derive $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$. Then we can also derive $\varepsilon \vdash P : \hat{\rho}_{m+1} \triangleright c'$ for some $c' \geq \log_2 c$.*

Lemma 5. *Suppose that $P \to_\beta Q$ is a β-reduction of order m, and we can derive $\Gamma \vdash Q : \hat{\tau} \triangleright c$ with $ord(\hat{\tau}) = m$. Then we can also derive $\Gamma \vdash P : \hat{\tau} \triangleright c$.*

Now the left-to-right implication of Theorem 2 easily follows. Indeed, take a closed term P of sort o and complexity m such that $\mathscr{L}(P)$ is infinite, and take any $c \in \mathbb{N}$. By $\log_2^k$ we denote the k-fold application of the logarithm: $\log_2^0 x = x$ and $\log_2^{k+1} x = \log_2(\log_2^k x)$. Since $\mathscr{L}(P)$ is infinite, it contains a tree t so big that $\log_2^m |t| \geq c$. We apply Lemma 3 to this tree, obtaining terms $Q_m, Q_{m-1}, \ldots, Q_0$ and a derivation of $\varepsilon \vdash Q_0 : \hat{\rho}_0 \triangleright |t|$. Then repeatedly for every $k \in \{1, \ldots, m\}$ we apply Lemma 4, obtaining a derivation of $\varepsilon \vdash Q_{k-1} : \hat{\rho}_k \triangleright c_k$ for some $c_k \geq \log_2^k |t|$, and Lemma 5 for every β-reduction (of order k) between Q_k and Q_{k-1}, obtaining a derivation of $\varepsilon \vdash Q_k : \hat{\rho}_k \triangleright c_k$. We end with a derivation of $\varepsilon \vdash P : \hat{\rho}_m \triangleright c_m$, where $c_m \geq \log_2^m |t| \geq c$, as needed. In the remaining part of this section we prove the three lemmata.

Proof of Lemma 3 (sketch). Recall that $t \in \mathscr{L}(P)$ is a finite tree, thus it can be found in some finite prefix of the Böhm tree of P. By definition, this prefix will be already expanded after performing some finite number of β-reductions from P. We need to observe that these β-reductions can be rearranged, so that those of higher order are performed first. The key point is to observe that when we perform a β-reduction of some order k, then no new β-redexes of higher order appear in the term. Indeed, suppose that $(\lambda x.R)\,S$ is changed into $R[S/x]$ somewhere in a term, where $ord(\lambda x.R) = k$. One new redex that may appear is when R starts with a λ, and to the whole $R[S/x]$ some argument is applied; this redex is of order $ord(R) \leq k$. Some other redexes may appear when S starts with a λ, and is substituted for such appearance of x to which some argument is applied; but this redex is of order $ord(S) < k$.

We can thus find a sequence of β-reduction in which β-reductions are arranged according to their order, that leads from P to some Q_0 such that t can be found in the prefix of Q_0 that is already expanded to a tree. It is now a routine to use the rules of our type system and derive $\varepsilon \vdash Q_0 : \hat{\rho}_0 \triangleright |t|$: in every br-labeled node we choose the subtree in which t continues, which effects in counting the number of nodes of t in the flag counter. □

Proof of Lemma 4. Consider some derivation of $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$. In this derivation we choose a leaf in which we will put the order-m marker, as follows. Starting from the root of the derivation, we repeatedly go to that premiss in which the flag counter is greatest (arbitrarily in the case of a tie). In every node that is not on the path to the selected leaf, we replace the current type judgement $\Gamma \vdash Q : (m, F, M, \tau) \triangleright d$ by $\Gamma \vdash Q : (m+1, F', M, \tau) \triangleright 0$, where $F' = F \cup \{m\}$ if $d > 0$, and $F' = F$ otherwise. In the selected leaf and all its ancestors, we change the order from m to $m+1$, we add m to the set of marker orders, and we recalculate the flag counter.

Let us see how such transformation changes the flag counter on the path to the selected leaf. We will prove (by induction) that the previous value d and the new value d' of the flag counter in every node on that path satisfy $d' \geq \log_2 d$. In the selected leaf itself, the flag counter (being either 0 or 1) remains unchanged; we have $d' = d \geq \log_2 d$. Next, consider any proper ancestor of the selected node. Let k be the number of those its children in which the flag counter was positive, plus the number of order-m flags placed in the considered node itself. Let also $d_{\max}$ and $d'_{\max}$ be the previous value and the new value of flag counter in this child that is in the direction of the selected leaf. By construction, the flag counter in this child was maximal, which implies $k \cdot d_{\max} \geq d$, while by the induction assumption $d'_{\max} \geq \log_2 d_{\max}$. To d' we take the flag counter only from the special child, while for other children with positive flag counter we add 1, i.e. $d' = k - 1 + d'_{\max}$. Altogether we obtain $d' = k - 1 + d'_{\max} \geq k - 1 + \log_2 d_{\max} \geq \log_2(k \cdot d_{\max}) \geq \log_2 d$, as required. □

Proof of Lemma 5. We consider the base case when $P = (\lambda x.R)\,S$ and $Q = R[S/x]$; the general situation (redex being deeper in P) is easily reduced to this one. In the derivation of $\Gamma \vdash Q : \hat{\tau} \triangleright c$ we identify the set I of places (nodes) where we derive a type for S substituted for x. For $i \in I$, let $\Sigma_i \vdash S : \hat{\sigma}_i \triangleright d_i$ be the type judgment in i. We change the nodes in I into leafs, where we instead derive $\varepsilon[x \mapsto \{\hat{\sigma}_i\}] \vdash x : \hat{\sigma}_i \triangleright 0$. It should be clear that we can repair the rest of the derivation, by changing type environments, replacing S by x in terms, and decreasing flag counters. In this way we obtain derivations of $\Sigma_i \vdash S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and a derivation of $\Sigma' \vdash R : \hat{\tau} \triangleright d$, where $\Sigma' = \Sigma[x \mapsto \{\hat{\sigma}_i \mid i \in I\}]$ with $\Sigma(x) = \emptyset$, and $Split(\Gamma \mid \Sigma, (\Sigma_i)_{i\in I})$, and $c = d + \Sigma_{i\in I} d_i$. To the latter type judgement we apply the (λ) rule, and then we merge it with the type judgements for S using the (@) rule, what results in a derivation for $\Gamma \vdash P : \hat{\tau} \triangleright c$. We remark that different $i \in I$ may give identical type judgements for S (as long as the set of markers in $\hat{\sigma}_i$ is empty); this is not a problem. The (@) rule requires that $ord(\hat{\sigma}_i) = ord(\lambda x.R)$; we have that $ord(\hat{\sigma}_i) = ord(\hat{\tau})$, and $ord(\hat{\tau}) = m = ord(\lambda x.R)$ by assumption. □

5 Soundness

In this section we sketch the proof of the right-to-left implication of Theorem 2. We, basically, need to reverse the proof from the previous section. The following new fact is now needed.

Lemma 6. *Suppose that we can derive* $\Gamma \vdash P : (m, F, M, \tau) \triangleright c$ *with* $m-1 \notin M$, *where* $ord(P) \leq m-1$ *and* $ord(x) \leq m-1$ *for all* x *with* $\Gamma(x) \neq \emptyset$. *Then* $c = 0$.

A simple inductive proof bases on the following idea: flags of order m are created only when a marker of order $m-1$ is visible; the derivation itself (together with free variables) does not provide it ($m-1 \notin M$), and the arguments, i.e. sets $T_1, \ldots, T_k$ in $\tau = T_1 \to \ldots \to T_k \to o$, may provide only markers of order at most $ord(P) - 1 \leq m-2$ (see the definition of a type), thus no flags of order m can be created.

We say that a term of the form PQ is an application *of order n* when $ord(P) = n$, and that an (@) rule is *of order n* if it derives a type for an application of order n. We can successively remove applications of the maximal order from a type derivation.

Lemma 7. *Suppose that* $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ *for* $m > 0$ *is derived by a derivation* D *in which the* (@) *rule of order* m *is used* n *times. Then there exists* Q *such that* $P \to_\beta^* Q$ *and* $\varepsilon \vdash Q : \hat{\rho}_m \triangleright c$ *can be derived by a derivation* D' *in which the* (@) *rule of order* m *is used less than* n *times.*

Recall from the definition of the type system that the (@) rule of orders higher than m cannot be used while deriving a full type of order m. Thus in D we have type judgements only for subterms of P of order at most m (although P may also have subterms of higher orders), and in type environments we only have variables of order at most $m-1$. In order to prove Lemma 7 we choose in P a subterm RS with $ord(R) = m$ such that there is a type judgement for RS in some nodes of D (at least one), but no descendants of those nodes use the (@) rule of order m. Since R is of order m, it cannot be an application (then we would choose it instead of RS) nor a variable; thus $R = \lambda x.R'$. We obtain Q by reducing the redex $(\lambda x.R')\,S$; the derivation D' is obtained by performing a surgery on D similar to that in the proof of Lemma 5 (but in the opposite direction). Notice that every full type (m, F, M, τ) (derived for S) with nonempty M is used for exactly one appearance of x in the derivation for R'; full types with empty M may be used many times, or not used at all, but thanks Lemma 6 duplicating or removing the corresponding derivations for S does not change the flag counter. In the derivations for $R'[S/x]$ no (@) rule of order m may appear, and the application RS disappears, so the total number of (@) rules of order m decreases.

When all (@) rules of order m are eliminated, we can decrease m.

Lemma 8. *Suppose that $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ for $m > 0$ is derived by a derivation D in which the (@) rule of order m is not used. Then we can also derive $\varepsilon \vdash P : \hat{\rho}_{m-1} \triangleright c'$ for some $c' \geq c$.*

The proof is easy; we simply decrease the order m of all derived full types by 1, and we ignore flags of order m and markers of order $m-1$. To obtain the inequality $c' \geq c$ we observe that when no (@) rule of order m is used, the information about flags of order $m-1$ goes only from descendants to ancestors, and thus every flag of order m is created because of a different flag of order $m-1$.

By repeatedly applying the two above lemmata, out of a derivation of $\varepsilon \vdash P : \hat{\rho}_m \triangleright c$ we obtain a derivation of $\varepsilon \vdash Q : \hat{\rho}_0 \triangleright c'$, where $P \to_\beta^* Q$ and $c' \geq c$. Using the latter derivation it is easy to find in the already expanded part of Q (and thus in $\mathscr{L}(Q) = \mathscr{L}(P)$) a tree t such that $|t| = c' \geq c$.

6 Conclusions

In this paper, we have shown an approach for expressing quantitative properties of Böhm trees using an intersection type system, on the example of the finiteness problem. It is an ongoing work to apply this approach to the diagonal problem, that should give a better complexity than that of the algorithm from [6]. Another ongoing work is to obtain an algorithm for model checking Böhm trees with respect to the Weak MSO+U logic [3]. This logic extends Weak MSO by a new quantifier U, expressing that a subformula holds for arbitrarily large finite sets. Furthermore, it seems feasible that our methods may help in proving a pumping lemma for nondeterministic HORSes.

References

[1] Achim Blumensath (2008): *On the Structure of Graphs in the Caucal Hierarchy*. *Theor. Comput. Sci.* 400(1-3), pp. 19–45, doi:10.1016/j.tcs.2008.01.053.

[2] Achim Blumensath (2013): *Erratum to "On the Structure of Graphs in the Caucal Hierarchy" [Theoret. Comput. Sci 400 (2008) 19-45]*. *Theor. Comput. Sci.* 475, pp. 126–127, doi:10.1016/j.tcs.2012.12.044.

[3] Mikołaj Bojańczyk & Szymon Toruńczyk (2012): *Weak MSO+U over Infinite Trees*. In Christoph Dürr & Thomas Wilke, editors: *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, *LIPIcs* 14, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 648–660, doi:10.4230/LIPIcs.STACS.2012.648.

[4] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague & Olivier Serre (2012): *A Saturation Method for Collapsible Pushdown Systems*. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts & Roger Wattenhofer, editors: *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, *Lecture Notes in Computer Science* 7392, Springer, pp. 165–176, doi:10.1007/978-3-642-31585-5_18.

[5] Christopher H. Broadbent & Naoki Kobayashi (2013): *Saturation-Based Model Checking of Higher-Order Recursion Schemes*. In Simona Ronchi Della Rocca, editor: *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, *LIPIcs* 23, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 129–148, doi:10.4230/LIPIcs.CSL.2013.129.

[6] Lorenzo Clemente, Paweł Parys, Sylvain Salvati & Igor Walukiewicz: *The Diagonal Problem for Higher-Order Recursion Schemes Is Decidable*. Submitted to LICS 2016.

[7] Lorenzo Clemente, Paweł Parys, Sylvain Salvati & Igor Walukiewicz (2015): *Ordered Tree-Pushdown Systems*. In Prahladh Harsha & G. Ramalingam, editors: *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, *LIPIcs* 45, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 163–177, doi:10.4230/LIPIcs.FSTTCS.2015.163.

[8] Werner Damm (1982): *The IO- and OI-Hierarchies*. *Theor. Comput. Sci.* 20, pp. 95–207, doi:10.1016/0304-3975(82)90009-3.

[9] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong & Olivier Serre (2008): *Collapsible Pushdown Automata and Recursion Schemes*. In: *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, IEEE Computer Society, pp. 452–461, doi:10.1109/LICS.2008.34.

[10] Alexander Kartzow & Paweł Parys (2012): *Strictness of the Collapsible Pushdown Hierarchy*. In Branislav Rovan, Vladimiro Sassone & Peter Widmayer, editors: *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, *Lecture Notes in Computer Science* 7464, Springer, pp. 566–577, doi:10.1007/978-3-642-32589-2_50.

[11] Teodor Knapik, Damian Niwiński & Paweł Urzyczyn (2002): *Higher-Order Pushdown Trees Are Easy*. In Mogens Nielsen & Uffe Engberg, editors: *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, *Lecture Notes in Computer Science* 2303, Springer, pp. 205–222, doi:10.1007/3-540-45931-6_15.

[12] Naoki Kobayashi (2009): *Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs*. In Zhong Shao & Benjamin C. Pierce, editors: *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, ACM, pp. 416–428, doi:10.1145/1480881.1480933.

[13] Naoki Kobayashi (2013): *Pumping by Typing*. In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, IEEE Computer Society, pp. 398–407, doi:10.1109/LICS.2013.46.

[14] Naoki Kobayashi, Kazuhiro Inaba & Takeshi Tsukada (2014): *Unsafe Order-2 Tree Languages Are Context-Sensitive*. In Anca Muscholl, editor: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, *Lecture Notes in Computer Science* 8412, Springer, pp. 149–163, doi:10.1007/978-3-642-54830-7_10.

[15] Naoki Kobayashi & C.-H. Luke Ong (2009): *A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes*. In: *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, IEEE Computer Society, pp. 179–188, doi:10.1109/LICS.2009.29.

[16] C.-H. Luke Ong (2006): *On Model-Checking Trees Generated by Higher-Order Recursion Schemes*. In: *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, IEEE Computer Society, pp. 81–90, doi:10.1109/LICS.2006.38.

[17] Paweł Parys (2012): *On the Significance of the Collapse Operation*. In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, IEEE Computer Society, pp. 521–530, doi:10.1109/LICS.2012.62.

[18] Paweł Parys (2014): *How Many Numbers Can a Lambda-Term Contain?* In Michael Codish & Eijiro Sumii, editors: *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, *Lecture Notes in Computer Science* 8475, Springer, pp. 302–318, doi:10.1007/978-3-319-07151-0_19.

[19] Steven J. Ramsay, Robin P. Neatherway & C.-H. Luke Ong (2014): *A Type-Directed Abstraction Refinement Approach to Higher-Order Model Checking*. In Suresh Jagannathan & Peter Sewell, editors: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, ACM, pp. 61–72, doi:10.1145/2535838.2535873.

[20] Sylvain Salvati & Igor Walukiewicz (2016): *Simply Typed Fixpoint Calculus and Collapsible Pushdown Automata*. *Mathematical Structures in Computer Science* FirstView, pp. 1–47, doi:10.1017/S0960129514000590.

A Examples

In this section we give few examples of derivations.

Example 1. Let us first analyze the term $P_1 = R(\lambda x.ax)$, where R is defined by coinduction as $R = (\lambda f.\mathsf{br}(fe)(R(\lambda x.f(fx))))$. Here a and e are symbols of rank 1 and 0, respectively. In $\mathscr{L}(P_1)$ there are trees that consist of a branch of a symbols ended with an e symbol, but only those where the number of a symbols is 2^k for some $k \in \mathbb{N}$. Notice that the complexity of P_1 is 2.

Let us first derive two full types for the subterm $\lambda x.ax$, namely

$$\hat{\tau}_{\mathsf{f}} = (2,\{1\},\emptyset,\{\hat{\rho}_1\}{\to}o), \qquad \text{and} \qquad \hat{\tau}_{\mathsf{m}} = (2,\emptyset,\{1\},\{\hat{\rho}_1\}{\to}o),$$

where we recall from page 6 that $\hat{\rho}_1 = (1,\emptyset,\{0\},o)$. Both $\hat{\tau}_{\mathsf{f}}$ and $\hat{\tau}_{\mathsf{m}}$ require an argument that provides the marker of order 0; $\hat{\tau}_{\mathsf{f}}$ provides a flag of order 1 and no markers, while $\hat{\tau}_{\mathsf{m}}$ provides the marker of order 1 and no flags. Denote $\varepsilon[x \mapsto \{\hat{\rho}_1\}] = \varepsilon[x \mapsto \{\hat{\rho}_1\}]$.

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0\},o)\triangleright 0}\,(\textsc{Var})}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash ax:(2,\{1\},\{0\},o)\triangleright 0}\,(\textsc{Con})}{\varepsilon\vdash \lambda x.ax:\hat{\tau}_{\mathsf{f}}\triangleright 0}\,(\lambda) \qquad \dfrac{\dfrac{\dfrac{}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0,1\},o)\triangleright 0}\,(\textsc{Var})}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash ax:(2,\emptyset,\{0,1\},o)\triangleright 1}\,(\textsc{Con})}{\varepsilon\vdash \lambda x.ax:\hat{\tau}_{\mathsf{m}}\triangleright 1}\,(\lambda)$$

In the derivation on the left, flags of order 0 and 1 are placed in the conclusion of the (Con) rule (flag of order 0 is created because we are in a constant; since the marker of order 0 is visible, we do not put 0 to the F set, but instead we create a flag of order 1). In the derivation on the right, the marker of order 1 is placed in the conclusion of the (Var) rule, which causes that this time in the conclusion of the (Con) rule flags of order 0, 1, and 2 are placed. Notice that in the conclusion of the (λ) rule (in both derivations) we remove 0 from the M set, because the order-0 marker is provided by x.

Next, we derive the full type $\hat{\sigma}_R = (2,\emptyset,\{0\},\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}{\to}o)$ for R:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash f:\hat{\tau}_{\mathsf{m}}\triangleright 0}\,(\textsc{Var}) \qquad \dfrac{}{\varepsilon\vdash e:(2,\{1\},\{0\},o)\triangleright 0}\,(\textsc{Con})}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash fe:(2,\emptyset,\{0,1\},o)\triangleright 1}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash \mathsf{br}(fe)(R(\lambda x.f(fx))):(2,\emptyset,\{0,1\},o)\triangleright 1}\,(\textsc{Br})}{\varepsilon\vdash R:\hat{\sigma}_R\triangleright 1}\,(\lambda)$$

In the conclusion of the (Con) rule we place the marker of order 0, and thus flags of order 0 and 1 are placed there. In the conclusion of the (@) rule the information about a flag of order 1 meets the information about the marker of order 1, and thus a flag of order 2 is placed, what increases the flag counter. Notice that the full type $\hat{\tau}_{\mathsf{f}}$ assigned to f in the type environment is never used; we can discard it because it provides no markers (it could be discarded also earlier, not necessarily in the leaf). On the other hand, the full type $\hat{\tau}_{\mathsf{m}}$ provides markers, so it could not be discarded (in particular, we could not pass it to the conclusion of the (Con) rule).

Next, we derive the same full type for R, but using the second argument of the br symbol; this results in greater values of the flag counter. We start by deriving the full type $\hat{\tau}_{\mathsf{f}}$ for the subterm $\lambda x.f(fx)$:

$$\dfrac{\dfrac{}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash f:\hat{\tau}_{\mathsf{f}}\triangleright 0} \qquad \dfrac{\dfrac{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash f:\hat{\tau}_{\mathsf{f}}\triangleright 0 \qquad \varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0\},o)\triangleright 0}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\},x\mapsto\{\hat{\rho}_1\}]\vdash fx:(2,\{1\},\{0\},o)\triangleright 0}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\},x\mapsto\{\hat{\rho}_1\}]\vdash f(fx):(2,\{1\},\{0\},o)\triangleright 0}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash \lambda x.f(fx):\hat{\tau}_{\mathsf{f}}\triangleright 0}\,(\lambda)$$

In the above derivation there are no flags nor markers. Next, we derive $\hat{\tau}_{\mathsf{m}}$ for the same subterm:

$$\dfrac{\dfrac{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash f:\hat{\tau}_{\mathsf{f}}\triangleright 0 \qquad \dfrac{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{m}}\}]\vdash f:\hat{\tau}_{\mathsf{m}}\triangleright 0 \qquad \varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0\},o)\triangleright 0}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{m}}\},x\mapsto\{\hat{\rho}_1\}]\vdash fx:(2,\emptyset,\{0,1\},o)\triangleright 0}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\},x\mapsto\{\hat{\rho}_1\}]\vdash f(fx):(2,\emptyset,\{0,1\},o)\triangleright 1}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash \lambda x.f(fx):\hat{\tau}_{\mathsf{m}}\triangleright 1}\,(\lambda)$$

Below the lower (@) rule the information about a flag of order 1 meets the information about the marker of order 1, and thus a flag of order 2 is placed, what increases the flag counter. We continue with the term R:

$$\dfrac{\dfrac{\dfrac{\varepsilon\vdash R:\hat{\sigma}_R\triangleright c \qquad \varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash \lambda x.f(fx):\hat{\tau}_{\mathsf{f}}\triangleright 0 \qquad \varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash \lambda x.f(fx):\hat{\tau}_{\mathsf{m}}\triangleright 1}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash R(\lambda x.f(fx)):(2,\emptyset,\{0,1\},o)\triangleright c+1}\,(@)}{\varepsilon[f\mapsto\{\hat{\tau}_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}]\vdash \mathsf{br}(fe)(R(\lambda x.f(fx))):(2,\emptyset,\{0,1\},o)\triangleright c+1}\,(\textsc{Br})}{\varepsilon\vdash R:\hat{\sigma}_R\triangleright c+1}\,(\lambda)$$

In this fragment of a derivation no flag nor counter is placed. In particular, there is no order-2 flag in conclusion of the (@) rule, although its second premiss provides a flag of order 1 while the third premiss provides the marker of order 1. We recall from the definition of the (@) rule that the information flags and markers coming the arguments is divided into two parts. Numbers smaller than than the order of the operator ($ord(R)=2$ in our case) are passed to the operator, while only greater numbers (≥ 2 in our case) contribute in creating new flags via the *Comp* predicate.

By composing the above fragments of a derivation, we can derive $\varepsilon\vdash R:\hat{\sigma}_R\triangleright c$ for every $c\geq 1$. Finally, we apply the argument, deriving the full type $\hat{\rho}_2$ for P_1.

$$\dfrac{\varepsilon\vdash R:\hat{\sigma}_R\triangleright c \qquad \varepsilon\vdash \lambda x.ax:\hat{\tau}_{\mathsf{f}}\triangleright 0 \qquad \varepsilon\vdash \lambda x.ax:\hat{\tau}_{\mathsf{m}}\triangleright 1}{\varepsilon\vdash P_1:\hat{\rho}_2\triangleright c+1}\,(@)$$

We can notice a correspondence between a derivation with flag counter $c+1$ and a tree in $\mathscr{L}(P)$ of size $2^{c-1}+1$. We remark that in every of these derivations only three flags of order 0 and only three flags of order 1 are present, in the three nodes using the (Con) rule.

Example 2. Consider a similar term $P_2=R(\lambda x.bxx)$, where R is as previously, and b is a symbol of rank 2. In $\mathscr{L}(P_2)$ we have, for every $k\in\mathbb{N}$, a full binary tree in which every branch consist of 2^k symbols b and ends with an e symbol.

This time for the subterm $\lambda x.bxx$ we need to derive three full types:

$$\begin{aligned}
&\hat{\tau}'_0=(2,\{0\},\emptyset,\{(1,\{0\},\emptyset,o)\}\to o),\\
&\hat{\tau}'_{\mathsf{f}}=(2,\{1\},\emptyset,\{(1,\{0\},\emptyset,o),\hat{\rho}_1\}\to o), \qquad \text{and}\\
&\hat{\tau}'_{\mathsf{m}}=(2,\emptyset,\{1\},\{(1,\{0\},\emptyset,o),\hat{\rho}_1\}\to o).
\end{aligned}$$

The last one is derived with flag counter 1. Notice that $\hat{\tau}'_{\mathsf{f}}$ and $\hat{\tau}'_{\mathsf{m}}$ need now two full types for the argument x; the new one $(1,\{0\},\emptyset,o)$ describes the subtree that is not on the path to the order-0 marker. We also have a new full type $\hat{\tau}'_0$ that describes the use of $\lambda x.bxx$ outside of the path to the order-0 marker.

Then, similarly as in the previous example, for every $c\geq 1$ we can derive $\varepsilon\vdash R:\hat{\sigma}'_R\triangleright c$, where $\hat{\sigma}'_R=(2,\emptyset,\{0\},\{\hat{\tau}'_0,\hat{\tau}'_{\mathsf{f}},\hat{\tau}'_{\mathsf{m}}\}\to o)$. Again, this allows to derive $\varepsilon\vdash P_2:\hat{\rho}_2\triangleright c+1$. This time a derivation with flag counter $c+1$ corresponds to a tree in $\mathscr{L}(P)$ of size 2^h-1 with $h=2^{c-1}+1$.

Example 3. This time consider the term $P_3 = R(\lambda x.x)$. The only tree in $\mathscr{L}(P_2)$ consists of a single e node. Let us see how the derivation from Example 1 has to be modified. The full type $\hat{\tau}_{\mathsf{m}}$ can still be derived for $\lambda x.x$ (although with flag counter 0 now), but instead of $\hat{\tau}_{\mathsf{f}}$ we have to use $\hat{\tau}''_{\mathsf{f}} = (2,\emptyset,\emptyset,\{\hat{\rho}_1\}\to o)$ that provides no flag of order 1:

$$\dfrac{\dfrac{}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0\},o)\triangleright 0}\,(\textsc{Var})}{\varepsilon\vdash\lambda x.x:\hat{\tau}''_{\mathsf{f}}\triangleright 0}\,(\lambda) \qquad \dfrac{\dfrac{}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0,1\},o)\triangleright 0}\,(\textsc{Var})}{\varepsilon\vdash\lambda x.x:\hat{\tau}_{\mathsf{m}}\triangleright 0}\,(\lambda)$$

Next, for R we want to derive the full type $\hat{\sigma}''_R = (2,\emptyset,\{0\},\{\hat{\tau}''_{\mathsf{f}},\hat{\tau}_{\mathsf{m}}\}\to o)$. We can easily adopt every of the previous derivations for $\varepsilon\vdash R:\hat{\sigma}_R\triangleright c$: we basically replace every $\hat{\tau}_{\mathsf{f}}$ by $\hat{\tau}''_{\mathsf{f}}$. The key point is that while deriving the full type $\hat{\tau}_{\mathsf{m}}$ for the subterm $\lambda x.f(fx)$, previously in the lower (@) rule we have received an information about an order-1 flag, and thus we have created an order-2 flag and increased the flag counter; this time there is no information about an order-1 flag, and thus we do not create an order-2 flag and do not increase the flag counter. In consequence, even if this part of the derivation is repeated arbitrarily many times, the value of the flag counter of the whole derivation remains 1.

Example 4. Finally, consider the term $P_4 = (\lambda g.P_3)(\lambda x.a(a(\dots(ax)\dots))$, that β-reduces to P_3. Notice that we can create the following derivation:

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{\dfrac{}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash x:(2,\emptyset,\{0\},o)\triangleright 0}\,(\textsc{Var})}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash ax:(2,\{1\},\{0\},o)\triangleright 0}\,(\textsc{Con})}{\varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash a(ax):(2,\{1\},\{0\},o)\triangleright 0}\,(\textsc{Con})\\ \vdots\\ \varepsilon[x\mapsto\{\hat{\rho}_1\}]\vdash a(a(\dots(ax)\dots)):(2,\{1\},\{0\},o)\triangleright 0\end{array}}{\varepsilon\vdash\lambda x.a(a(\dots(ax)\dots)):\hat{\tau}_{\mathsf{f}}\triangleright 0}\,(\textsc{Con})\,(\lambda)$$

Every (Con) rule used in this derivation places in its conclusion an order-0 flag and an order-1 flag. This derivation can be used as a part of a derivation for P_4:

$$\dfrac{\dfrac{\varepsilon[g\mapsto\{\hat{\tau}_{\mathsf{f}}\}]\vdash P_3:\hat{\rho}_2\triangleright 1}{\varepsilon\vdash\lambda g.P_3:(2,\emptyset,\{0,1\},\{\hat{\tau}_{\mathsf{f}}\}\to o)\triangleright 1}\,(\lambda) \qquad \varepsilon\vdash\lambda x.a(a(\dots(ax)\dots)):\hat{\tau}_{\mathsf{f}}\triangleright 0}{\varepsilon\vdash P_4:\hat{\rho}_2\triangleright 1}\,(@)$$

Because $\hat{\tau}_{\mathsf{f}}$ provides no markers, it can be removed from the type environment and thus for P_3 we can use the derivation from the previous example. We thus obtain a derivation for P_4 in which there are many order-0 and order-1 flags (but only one flag of order 2). This shows that in the flag counter we indeed need to count only the number of flags of the maximal order (not, say, the total number of flags of all orders).