A Type System Describing Unboundedness

Pawel Parys

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **22**:4, 2020, #2

A Type System Describing Unboundedness[*]

Paweł Parys

Institute of Informatics, University of Warsaw, Poland

received 2018-08-13, *revised 2019-11-05, 2020-07-30*, *accepted 2020-08-03*.

We consider nondeterministic higher-order recursion schemes as recognizers of languages of finite words or finite trees. We propose a type system that allows to solve the simultaneous-unboundedness problem (SUP) for schemes, which asks, given a set of letters A and a scheme $\mathcal{G}$, whether it is the case that for every number n the scheme accepts a word (a tree) in which every letter from A appears at least n times. Using this type system we prove that SUP is $(m-1)$-EXPTIME-complete for word-recognizing schemes of order m, and m-EXPTIME-complete for tree-recognizing schemes of order m. Moreover, we establish the reflection property for SUP: out of an input scheme $\mathcal{G}$ one can create its enhanced version that recognizes the same language but is aware of the answer to SUP.

Keywords: simultaneous-unboundedness problem, higher-order recursion schemes, intersection types, reflection

1 Introduction

The *simultaneous-unboundedness problem* (*SUP* for short, also known as the *diagonal problem*) in its original formulation over finite words asks, for a set of letters A and a language of words L, whether for every $n \in \mathbb{N}$ there is a word in L where every letter from A occurs at least n times. The same problem can be also considered for a language of finite trees. In this paper, we are interested in solving SUP for languages of finite words and finite trees recognized by nondeterministic higher-order recursion schemes.

Higher-order recursion schemes (*schemes* in short) are used to faithfully represent the control flow of programs in languages with higher-order functions. This formalism is equivalent via direct translations to simply-typed λY-calculus (Salvati and Walukiewicz, 2016) and to higher-order OI grammars (Damm, 1982; Kobele and Salvati, 2015). Collapsible pushdown systems (Hague, Murawski, Ong, and Serre, 2008) and ordered tree-pushdown systems (Clemente, Parys, Salvati, and Walukiewicz, 2015) are other equivalent formalisms. Schemes cover some other models such as indexed grammars (Aho, 1968) and ordered multi-pushdown automata (Breveglieri, Cherubini, Citrini, and Crespi-Reghizzi, 1996).

By a recent result by Clemente, Parys, Salvati, and Walukiewicz (2016) we know that SUP for higher-order recursion schemes is decidable. For schemes of order m their algorithm works in $f(m)$-fold exponential time for some quadratic function f (although the complexity of the algorithm is not mentioned explicitly in the paper, it can be easily estimated as being such). Their solution is based on two transformations that simplify a scheme without changing the answer to SUP. These transformations, repeated alternatingly, reduce the input scheme to a scheme of order 0, for which solving SUP becomes trivial.

[*]Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

We analyze SUP for schemes using an appropriate system of intersection types. Intersection types were intensively used in the context of schemes, for several purposes like model-checking (Kobayashi, 2009; Kobayashi and Ong, 2009; Broadbent and Kobayashi, 2013; Ramsay, Neatherway, and Ong, 2014), pumping (Kobayashi, 2013), transformations of schemes (Kobayashi, Inaba, and Tsukada, 2014; Asada and Kobayashi, 2016; Clemente et al., 2016), and so on. Among such type systems we have to distinguish those (Parys, 2016, 2017b), in which the (appropriately defined) size of a type derivation for a term approximates some quantity visible in the Böhm tree of that term. In particular, in our recent work (Parys, 2017b) we have developed a type system that allows to solve SUP for a special case of a single-letter alphabet.

Here, we generalize the last type system mentioned above to multiple letters. As a result, a type derivation in this system is labeled by flags of different kinds. The key property lies in some (quite rough) correspondence between words (trees) that can be generated from a term and type derivations for the term, where, for every letter a, the number of occurrences of a in the generated word (tree) is approximated by the number of occurrences of an appropriate flag in the type derivation. In consequence, SUP reduces to checking whether there exist type derivations with arbitrarily many flags corresponding to every letter from the input set A.

Thanks to a careful optimization of the developed type system, we obtain an algorithm solving SUP in the optimal complexity. Namely, we prove that SUP is $(m-1)$-EXPTIME-complete for word-recognizing schemes of order m, and m-EXPTIME-complete for tree-recognizing schemes of order m. The corresponding lower bounds are obtained effortlessly, because already much simpler problems for schemes require such a complexity.

Let us recall from Clemente et al. (2016) that the decidability result for SUP entailed other decidability results for recursion schemes, concerning in particular computability of the downward closure of recognized languages (Zetzsche, 2015), and the problem of separability by piecewise testable languages (Czerwiński, Martens, van Rooijen, and Zeitoun, 2015). Although our complexity result for SUP does not influence directly our knowledge on the complexity of the other problems (the aforementioned reductions preserve only decidability, but not complexity), it can be seen as the first step in establishing the complexity of the other problems as well.

Going further, we constitute the reflection property for SUP: out of an input scheme $\mathcal{G}$ one can create its enhanced version that recognizes the same language, but in every moment of the recognition process it is aware of the answer to SUP. In order to obtain this result, we adapt a construction of Haddad (2012, Section 4.2) to the setting of our type system. The reflection property for SUP allows to solve SUP simultaneously for infinitely many languages at once, if they are all described by a single scheme. This has been already used in our recent paper (Parys, 2018b) to establish decidability of model-checking trees generated by recursion schemes with respect to the WMSO+U logic.

The current paper is a full version of a conference paper (Parys, 2017a). The part about the reflection property (Section 10) comes from another conference paper (Parys, 2018b).

Our paper is structured as follows. In Section 2 we introduce all necessary definitions. In Section 3 we introduce the type system describing simultaneous unboundedness for word-recognizing schemes. Sections 4-7 are devoted to a proof of correctness of the type system. In Section 8 we present a translation from tree-recognizing schemes to word-recognizing schemes. Next, in Section 9 we show how the type system can be used to solve SUP. Then, in Section 10 we establish the reflection property for SUP. Finally, in Section 11 we comment on relations to the task of computing downward closures of languages.

2 Preliminaries

Infinitary lambda-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. We omit brackets on the right of an arrow, so e.g. $o \to (o \to o)$ is abbreviated to $o \to o \to o$. The order of a sort is defined by induction: $ord(o) = 0$, and $ord(\alpha_1 \to \ldots \to \alpha_s \to o) = 1 + \max(ord(\alpha_1), \ldots, ord(\alpha_s))$ for $s \geq 1$.

A sort $\alpha_1 \to \ldots \to \alpha_s \to o$ is *homogeneous* if $ord(\alpha_1) \geq \cdots \geq ord(\alpha_s)$ and all $\alpha_1, \ldots, \alpha_s$ are homogeneous. In the sequel we restrict ourselves to homogeneous sorts (even if not always this is written explicitly).

Let Σ be a set of letters (alphabet). We assume that Σ is finite. To denote nondeterministic choices we use a symbol nd. Assuming that $\mathsf{nd} \notin \Sigma$, we denote $\Sigma^{\mathsf{nd}} = \Sigma \cup \{\mathsf{nd}\}$. Let also $Vars = \{x^\alpha, y^\beta, z^\gamma, \ldots\}$ be a set of variables, containing infinitely many variables of every homogeneous sort (sort of a variable is written in superscript).

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

- node constructor—if $a \in \Sigma^{\mathsf{nd}}$, and $P_1^o, \ldots, P_r^o$ are lambda-terms, then $(a\langle P_1^o, \ldots, P_r^o\rangle)^o$ is a lambda-term,
- variable—every variable $x^\alpha \in Vars$ is a lambda-term,
- application—if $P^{\alpha \to \beta}$ and Q^α are lambda-terms, then $(P^{\alpha \to \beta}\, Q^\alpha)^\beta$ is a lambda-term, and
- lambda-binder—if P^β is a lambda-term and x^α is a variable, then $(\lambda x^\alpha.P^\beta)^{\alpha \to \beta}$ is a lambda-term;

in the above, α, β, and $\alpha \to \beta$ are homogeneous sorts. We naturally identify lambda-terms differing only in names of bound variables. We often omit the sort annotations of lambda-terms, but we keep in mind that every lambda-term (and every variable) has a particular sort. *Substitution*, *beta-reduction*, and *free variables* of a lambda-term are defined as usual. A lambda-term P is *closed* if it has no free variables. We write $\xrightarrow{h}_\beta$ for a *head beta-reduction* defined as follows: we have $P \xrightarrow{h}_\beta Q$ if $P = (\lambda x.R)\, S\, S_1\, \ldots\, S_s$ and $Q = R[S/x]\, S_1\, \ldots\, S_s$.

For a lambda-term P, the *order* of P is just the order of its sort, while the *complexity* of P is the smallest number m such that the order of all subterms of P is at most m. We restrict ourselves to lambda-terms that have finite complexity. We also define the order of a beta-reduction as the order of the involved variable. More precisely, for a number $k \in \mathbb{N}$, we say that there is a beta-reduction of order k from a lambda-term P to a lambda-term Q, written $P \to_{\beta(k)} Q$, if Q is obtainable from P by replacing a redex $(\lambda x.R)\, S$ where $ord(x) = k$ with $R[S/x]$.

Trees. A *tree* is defined as a lambda-term that is built using only node constructors, that is, not using variables, applications, nor lambda-binders. A tree is Γ-*labeled* if only letters from Γ appear in it.

Let us now define how we resolve nondeterministic choices. Although this is mainly used for trees, we define it for arbitrary lambda-terms. We write $P \to_{\mathsf{nd}} Q$ if Q is obtained from P by choosing some occurrence of the nd symbol surrounded only by node constructors with letters from Σ, and removing this nd symbol together with all but one of its arguments. Formally, we let $\to_{\mathsf{nd}}$ to be the smallest relation such that $\mathsf{nd}\langle P_1, \ldots, P_r\rangle \to_{\mathsf{nd}} P_i$ for $i \in \{1, \ldots, r\}$, and if $a \in \Sigma$, and $P_i \to_{\mathsf{nd}} P_i'$ for some $i \in \{1, \ldots, r\}$, and $P_j = P_j'$ for all $j \in \{1, \ldots, r\} \setminus \{i\}$, then $a\langle P_1, \ldots, P_r\rangle \to_{\mathsf{nd}} a\langle P_1', \ldots, P_r'\rangle$. For a relation r, by r^* we denote the reflexive transitive closure of r. For a lambda-term P (which is usually a Σ^{nd}-labeled, potentially infinite tree), by $\mathcal{L}(P)$ we denote the set of all finite, Σ-labeled trees T such that $P \to_{\mathsf{nd}}^* T$.

Böhm trees. We consider Böhm trees only for closed lambda-terms of sort o. For such a term P, its *Böhm tree* $BT(P)$ is constructed by coinduction, as follows: if $P \xrightarrow{h}_\beta^* a\langle P_1, \dots, P_r\rangle$ (for some $a \in \Sigma^{\mathsf{nd}}$ and some lambda-terms $P_1, \dots, P_r$), then $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$; otherwise $BT(P) = \mathsf{nd}\langle\rangle$.

Notice that for every lambda-term P there exists at most one Q such that $P \xrightarrow{h}_\beta Q$, and that if P is already of the form $a\langle P_1, \dots, P_r\rangle$ then $P \xrightarrow{h}_\beta Q$ does not hold for any Q. In consequence, there is at most one lambda-term Q of the form $a\langle P_1, \dots, P_r\rangle$ for which $P \xrightarrow{h}_\beta^* Q$, and thus the Böhm tree of every lambda-term is uniquely defined.

Due to the standardization theorem, for every closed lambda-term P of sort o, it is the case that $BT(P) = BT(Q)$ whenever $P \to_\beta Q$; in particular, if $P \to_\beta^* a\langle P_1, \dots, P_r\rangle$, then $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$.

All finite lambda-terms P are strongly normalizing: every maximal sequence of beta-reductions starting in P is finite and ends in the same lambda-term, called the *beta-normal form* of P. Additionally, if P is finite, closed, and of sort o, then necessarily $BT(P)$ equals the beta-normal form of P (this is the case, because for every finite closed lambda-term Q of sort o either Q starts with a node constructor, or we have $Q \xrightarrow{h}_\beta R$ for some R).

Higher-order recursion schemes. We use a very loose definition of schemes. A *higher-order recursion scheme* (or just a *scheme*) is a triple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^o)$, where $\mathcal{N} \subseteq \mathit{Vars}$ is a finite set of nonterminals, $\mathcal{R}$ is a function that maps every nonterminal $N \in \mathcal{N}$ to a finite lambda-term whose free variables are contained in $\mathcal{N}$ and whose sort equals the sort of N, and $N_0^o \in \mathcal{N}$ is a starting nonterminal, being of sort o. We assume that elements of $\mathcal{N}$ are not used as bound variables, and that $\mathcal{R}(N)$ is not a nonterminal. We sometimes say that $\mathcal{R}$ defines *rules* of the scheme. The order of the scheme is defined as the maximum of complexities of $\mathcal{R}(N)$ over all its nonterminals N.

For a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^o)$, and for a lambda-term P (possibly containing some nonterminals from $\mathcal{N}$), let $\Lambda_\mathcal{G}(P)$ be the lambda-term obtained as a limit of applying recursively the following operation to P: take an occurrence of some nonterminal N, and replace it by $\mathcal{R}(N)$ (the nonterminals should be chosen so that every nonterminal is eventually replaced). We remark that while substituting $\mathcal{R}(N)$ for a nonterminal N, there is no need for any renaming of variables (capture-avoiding substitution), since $\mathcal{R}(N)$ does not have free variables other than nonterminals. The infinitary lambda-term *represented by* $\mathcal{G}$ is defined as $\Lambda_\mathcal{G}(N_0^o)$, and is denoted $\Lambda(\mathcal{G})$. Observe that $\Lambda(\mathcal{G})$ is a closed lambda-term of sort o and of complexity not greater than the order of the scheme. The *language of* $\mathcal{G}$ is defined as $\mathcal{L}(\mathcal{G}) = \mathcal{L}(BT(\Lambda(\mathcal{G})))$.

We remark that according to our definition all subterms of all lambda-terms (and all nonterminals as well) have homogeneous sorts; usually it is not assumed that sorts used in a scheme are homogeneous. It is, however, the case that any scheme using also non-homogeneous sorts can be converted into one in which all sorts are homogeneous, and that this can be done in logarithmic space (Parys, 2018a). We make the homogeneity assumption for technical convenience. We refer the reader to Appendix A for a comment on other differences between our definition and the usual one.

A *word* is defined as a tree in which every node has at most one child (such a tree can be identified with a word understood in the classic sense). We say that a closed lambda-term P of sort o (or a scheme $\mathcal{G}$) is *word-recognizing* if all elements of $\mathcal{L}(BT(P))$ (or $\mathcal{L}(\mathcal{G})$, respectively) are words.

Example 2.1. Consider the higher-order recursion scheme $\mathcal{G}_1$ with two nonterminals, M^o (taken as a starting nonterminal) and $\mathsf{N}^{(o\to o)\to o}$, and with rules

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle)\,, \qquad \mathcal{R}(\mathsf{N}) = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), \mathsf{N}\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle\,.$$

This recursion scheme is of order 2. We obtain $\Lambda(\mathcal{G}_1) = R_1\,(\lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle)$, where R_1 is the unique lambda-term such that $R_1 = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. We have $BT(\Lambda(\mathcal{G}_1)) = \mathsf{nd}\langle T_{2^0}, \mathsf{nd}\langle T_{2^1}, \mathsf{nd}\langle T_{2^2}, \ldots\rangle\rangle\rangle$, where $T_0 = \mathsf{c}\langle\rangle$ and $T_{i+1} = \mathsf{nd}\langle \mathsf{a}\langle T_i\rangle, \mathsf{b}\langle T_i\rangle\rangle$. In $\mathcal{L}(\mathcal{G}_1)$ we have words of length $2^i + 1$ for all $i \in \mathbb{N}$, where the first 2^i letters are chosen from $\{\mathsf{a}, \mathsf{b}\}$ arbitrarily, and the last letter is c. In the following examples we continue to consider this scheme, assuming that $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$; using our type system, we want to exhibit the fact that in $\mathcal{L}(\mathcal{G}_1)$ there are words having simultaneously many letters a and many letters b. □

Let us also define formally the size of a higher-order recursion scheme. The size of a sort α, denoted $|\alpha|$, is defined by induction on the structure of α: $|o| = 1$ and $|\alpha \to \beta| = |\alpha| + 1 + |\beta|$. The size of a finite lambda-term P, denoted $|P|$, is also defined by induction on its structure, as follows:

$$|a\langle P_1, \ldots, P_r\rangle| = 1 + |P_1| + \cdots + |P_r|\,, \qquad |P\,Q| = |P| + 1 + |Q|\,,$$
$$|x^\alpha| = 1\,, \qquad |\lambda x^\alpha.P| = |\alpha| + 1 + |P|\,.$$

Finally, the size of a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$, denoted $|\mathcal{G}|$, is defined as

$$|\mathcal{G}| = \sum_{N^\alpha \in \mathcal{N}} (|\alpha| + |\mathcal{R}(N^\alpha)|)\,.$$

Remark. We notice that in the size of a scheme we include sizes of sorts of all bound variables and all nonterminals. Although in "reasonable cases" lambda-terms using large sorts are large anyway, this is sometimes important. For example, in the size of the lambda-term $(\lambda \mathsf{x}^{\alpha \to o}.\mathsf{a}\langle\rangle)\,(\lambda \mathsf{y}^\alpha.\mathsf{a}\langle\rangle)$ we prefer to include the size of α, as otherwise the size of this lambda-term would be small even for a very large α. Similarly, in $\mathcal{G}$ we can have a nonterminal N of some sort α with a rule $\mathcal{R}(\mathsf{N}) = (\lambda \mathsf{x}^o.\mathsf{N})\,(\mathsf{a}\langle\rangle)$; in such a case we also prefer to include the size of α in the size of $\mathcal{G}$.

3 Type system for simultaneous unboundedness

In this section we introduce a type system that allows to solve SUP for word-recognizing schemes. SUP for tree-recognizing schemes is solved in Section 8 by a reduction to SUP for word-recognizing schemes.

Definition 3.1. For a set of trees L and a set of letters A, the predicate $SUP_A(L)$ holds if for every $n \in \mathbb{N}$ there is some $T \in L$ with at least n occurrences of every letter from A. The *simultaneous-unboundedness problem* (*SUP*) for tree-recognizing order-m schemes is to decide whether $SUP_A(\mathcal{L}(\mathcal{G}))$ holds, given a scheme $\mathcal{G}$ of order at most m and a set A. SUP for *word-recognizing* order-m schemes is as the above, but with the restriction that $\mathcal{G}$ is word-recognizing.

We remark that the language of infinite trees T for which $SUP_A(\mathcal{L}(T))$ holds is not regular (due to a simple pumping argument)—SUP talks about unboundedness of some quantities. This makes the problem inaccessible to standard methods used for analyzing schemes, as they usually concern only regular properties of the Böhm tree; it was necessary to develop methods accessing some quantities visible in the Böhm tree.

Intuitions. The main novelty of our type system lies in labeling nodes of type derivations by two kinds of labels called flags and markers. To each marker we assign a number, called an order. Flags, besides their order, are also identified by a letter from Σ; thus we have (k, a)-flags for $k \geq 1$ and $a \in \Sigma$. While deriving

a type for a lambda-term of complexity at most $m+1$ (i.e., where every variable has order at most m), we use markers of order from the range $0, \ldots, m$, and flags of order from the range $1, \ldots, m+1$.

Let P_{m+1} be a word-recognizing closed lambda-term of sort o and of complexity at most $m+1$. Recall that our goal is to describe a word $T \in \mathcal{L}(BT(P_{m+1}))$ using a type derivation for P_{m+1} itself. While doing that, we want to preserve information that T has many occurrences of every letter from a set A.

Since T can be found in some finite prefix of $BT(P_{m+1})$, in order to find T it is enough to perform finitely many beta-reductions from P_{m+1}. Moreover, thanks to the fact that all sorts are homogeneous, the beta-reductions can be rearranged so that those of higher order are performed first (as shown in Lemma 6.1). Namely, we can find lambda-terms $P_0, \ldots, P_m$ such that

$$P_{m+1} \to^*_{\beta(m)} P_m \to^*_{\beta(m-1)} \cdots \to^*_{\beta(0)} P_0 \qquad \text{and} \qquad P_0 \to^*_{\mathsf{nd}} T\,.$$

Recall that our goal is to identify places of P_{m+1} responsible for creating nodes of T, and label them somehow using flags and markers. In order to achieve this, we start by labeling P_0, and then we transfer the labels back to P_1, P_2, and so on.

Some prefix of P_0 can be seen as a tree, in which we can find all nodes of T, interleaved with some additional nd-labeled nodes. Let us place a marker of order 0 in the leaf of P_0 that is taken as the (unique) leaf of T. Additionally, for every letter $a \in \Sigma$, let us place $(1,a)$-flags in all a-labeled nodes of P_0 that are taken to T.

Notice that every node constructor of P_0 was created out of some particular occurrence of a node constructor in P_1. Using this correspondence, we move flags from P_0 back to P_1. Namely, we find node constructors of P_1 out of which in P_0 we obtain node constructors labeled by $(1,a)$-flags, and we put $(1,a)$-flags also in these node constructors of P_1; similarly we proceed with the marker of order 0. The crucial observation is that no two flagged node constructors of P_0 could come out of a single node constructor of P_1. Indeed, recall that all the beta-reductions between P_1 and P_0 are of order 0. This means that in every such beta-reduction we take a whole subtree (i.e., a lambda-term of sort o) of P_1, and we replace it somewhere, possibly replicating it. But since all $(1,a)$-flags lie in P_0 on a single path, they may lie only in at most one copy of the replicated subtree.

Next, we would like to move flags from P_1 back to P_2, then to P_3, and so on, so that the number of occurrences of particular letters in T would be reflected in the number of flags in the original lambda-term P_{m+1}. We cannot do this directly, though. The problem is that flags in P_1 (unlike in P_0) do not need to be located on a single path, and, in consequence, a single node constructor in P_2 may result in multiple (uncontrollably many) node constructors with a flag in P_1. We rescue ourselves by considering only $|\Sigma|$ paths in P_1, one for each letter in Σ. Namely, for every letter $a \in \Sigma$ we place in P_1 a marker of order 1, choosing in this way the path from the root to the position of this marker. Then, for every node labeled by a $(1,a)$-flag we place a $(2,a)$-flag in the closest ancestor that lies on the chosen path. Although the number of $(2,a)$-flags may be smaller than the number of $(1,a)$-flags (the closest ancestor on the path may be the same for multiple $(1,a)$-flags), we can ensure that it is smaller only logarithmically; to do so, we choose the marked node in a clever way: starting from the root, we always proceed to this subtree in which the number of $(1,a)$-flags is the largest. In consequence, if the number of $(1,a)$-flags was "very large", then also the number of $(2,a)$-flags remains "very large".

After this additional step, we transfer all flags and markers from P_1 to P_2 (i.e., whenever a node constructor of P_1 is labeled by a flag or a marker, we put the same label in the corresponding node constructor of P_2). Since for every $a \in \Sigma$ all $(2,a)$-flags lie on a single path of P_1, the number of

occurrences of order-2 flags (for every letter a) is the same in P_2 as in P_1. Although the number of occurrences of order-1 flags may change, we transfer them to P_2 as well; thanks to their presence, we are able to easily check correctness of the labeling of P_2 by order-2 flags.

We continue by repeating the same process until reaching P_{m+1}: in P_2 we again reduce our considerations to $|\Sigma|$ paths by introducing markers of order 2, we place $(3, a)$-flags on these paths, we transfer all flags and markers back to P_3, and so on. At the end we obtain some labeling of P_{m+1} by several kinds of flags and markers. The goal of the type system we develop is, roughly speaking, to ensure that a labeling of P_{m+1} by flags and markers is actually obtainable in the process as above (in fact, we are not labeling nodes of P_{m+1} itself, but rather nodes of a type derivation for P_{m+1}).

Example 3.1. Let us take $P_2 = \Lambda(\mathcal{G}_1)$, where $\mathcal{G}_1$ is the scheme from Example 2.1 (the complexity of P_2 is indeed 2). In $\mathcal{L}(BT(P_2))$ we have a word $T = \mathsf{a}\langle\mathsf{a}\langle\mathsf{a}\langle\mathsf{a}\langle\mathsf{c}\langle\rangle\rangle\rangle\rangle\rangle$. Below we recall the shape of P_2, replacing by $\Box$ subterms irrelevant for generating T (the vertical dots should be ignored for now, their meaning is explained later):

$$P_2 = \Big(\lambda\mathsf{f}.\mathsf{nd}\,\Big\langle \Box, \Big(\lambda\mathsf{f}.\mathsf{nd}\Big\langle \Box, \big(\lambda\mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), \Box\rangle\big)\,\big(\lambda\mathsf{y}'.\mathsf{f} \vdots (\mathsf{f}\,\mathsf{y}')\big)\Big\rangle\Big)\,\big(\lambda\mathsf{y}.\mathsf{f} \vdots (\mathsf{f} \vdots \mathsf{y})\big)\Big\rangle\Big) \big(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle\big)\,.$$

After performing some β-reductions of order 1, we obtain the lambda-term

$$P_1 = \mathsf{nd}\Big\langle \Box, \mathsf{nd}\Big\langle \Box, \mathsf{nd}\Big\langle \Big(\lambda\mathsf{y}'.\Big(\lambda\mathsf{y}.\big(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle\big)\,\Big(\big(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle\big)\,\mathsf{y}\Big)\Big) \vdots \Big(\Big(\lambda\mathsf{y}.\big(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle\big) \vdots \Big(\big(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle\big) \vdots \underline{\mathsf{y}}\Big)\Big)\,\mathsf{y}'\Big)\Big)\,(\mathsf{c}\langle\rangle), \Box\Big\rangle\Big\rangle\Big\rangle\,.$$

We remark that in the part of P_1 hidden under $\Box$ many variables of order 1 remain unreduced (we have performed only a finite number of beta-reductions). Then, after performing also β-reductions of order 0, we obtain the lambda-term

$$P_0 = \mathsf{nd}\big\langle \Box, \mathsf{nd}\big\langle \Box, \mathsf{nd}\big\langle \mathsf{nd}\langle\mathsf{a}\langle\mathsf{nd}\langle\mathsf{a}\langle\mathsf{nd}\langle\mathsf{a}\langle\mathsf{nd}\langle\mathsf{a}\langle\mathsf{c}\langle\rangle\rangle, \Box\rangle\rangle, \Box\rangle\rangle, \Box\rangle\rangle, \Box\rangle, \Box\big\rangle\big\rangle\big\rangle\,,$$

and we have $P_0 \to^*_{\mathsf{nd}} T$.

The four a-labeled node-constructors of T are present also in P_0; we label them by $(1, \mathsf{a})$-flags. We see that in P_0 they are located on a single path. Each of them originates from a different node-constructor in P_1; in P_1 we again see the four node-constructors, and we label them by $(1, \mathsf{a})$-flags. In P_1, however, they are no longer located on a path (we see that they appear in four independent occurrences of $\mathsf{a}\langle\mathsf{x}\rangle$). We then see that the four node-constructors in P_1 originate from a single node-constructor in P_2. This node-constructor in P_2 becomes again labeled by a $(1, \mathsf{a})$-flag, but the number of $(1, \mathsf{a})$-flags in P_2 is "far from" their number in P_0. It is thus indeed necessary to do something else, that is, to place in P_1 a marker of order 1 and some $(2, \mathsf{a})$-flags. We place the marker in the underlined occurrence of the variable y. Then, for every node-constructor containing a $(1, \mathsf{a})$-flag we look for the closest common ancestor with the occurrence of y containing the marker, and we place a $(2, \mathsf{a})$-flag in this ancestors. The ancestors are "application nodes" of the lambda-term. For the last a, this is the rightmost place denoted in P_1 by vertical dots (i.e., the place where as an argument to $\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \Box\rangle$ we apply y). For the previous a, this is the

second place denoted in P_1 by vertical dots. For the first two a's, this is the first place denoted in P_1 by vertical dots. Thus, in P_1 we have three $(2,\mathsf{a})$-flags. Notice that they are located on a single path. Each of the three places denoted in P_1 by vertical dots originates from a different place in P_2 (also denoted there by vertical dots). In consequence, in P_2 we also have three $(2,\mathsf{a})$-flags. □

Type judgments. For storing information about flags and markers used in a derivation of a type we use flag sets and marker multisets. Recall that a flag is parameterized by a pair (k,a), where $k \geq 1$ is called an order, and $a \in \Sigma$ is called a letter. For flags it is enough to remember for every order whether at least one flag of this order was used, and if so, then also a letter of this flag (if flags with multiple letters were used, it is enough for us to remember just one of these letters). Thus for $m \geq 0$ we define

$$\mathcal{F}_m = \big\{F \subseteq \{1,\dots,m\} \times \Sigma \mid (k,a),(k,b) \in F \Rightarrow a = b\big\}.$$

Sets F in $\mathcal{F}_m$ are called *m-bounded flag sets*. For markers the situation is slightly different, as we want to remember precisely how many markers were used. Moreover, markers do not have a letter, only an order. We thus define

$$\mathcal{M}_m = \big\{M\colon \mathbb{N} \to \{0,\dots,|\Sigma|\} \mid M(0) \leq 1 \wedge \forall k > m.\, M(k) = 0\big\}.$$

Functions M in $\mathcal{M}_m$ are called *m-bounded marker multisets*. The intention is that $M(k)$ says how many markers of order k were used.

By $M+M'$ and $M-M'$ we mean the coordinatewise sum or difference, respectively. We use $\mathbf{0}$ to denote a function that maps every element of its domain to 0 (where the domain should be always clear from the context). By $\{\!\{k_1,\dots,k_n\}\!\}$ we mean the multiset M such that $M(k) = |\{i \in \{1,\dots,n\} \mid k_i = k\}|$ for all $k \in \mathbb{N}$. When $F \in \mathcal{F}_m$, $M \in \mathcal{M}_m$, $n \in \mathbb{N}$, and $\Box$ is one of $\leq, >$, we write $F{\restriction_{\Box n}}$ for $\{(k,a) \in F \mid k \Box n\}$, and $M{\restriction_{\Box n}}$ for the function that maps every k to $M(k)$ if $k \Box n$, and to 0 if $\neg(k \Box n)$.

Next, for every sort α and for $m \geq 0$ we define three sets: the set $\mathcal{T}^\alpha$ of *types* of sort α, the set $\mathcal{TT}^\alpha_m$ of *m-bounded type triples* of sort α, and the set $\mathcal{TC}^\alpha$ of *triple containers* of sort α. They are defined by mutual induction on the structure of α.

For $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o$ we define a set of *types* as follows:

$$\mathcal{T}^\alpha = \mathcal{TC}^{\alpha_1} \times \cdots \times \mathcal{TC}^{\alpha_s}.$$

A type $(C_1,\dots,C_s) \in \mathcal{T}^\alpha$ is written in the form $C_1 \to \ldots \to C_s \to o$.

Then, we define (using $\mathsf{Mk}(C_i)$ defined below)

$$\mathcal{TT}^\alpha_m = \big\{(F,M,C_1 \to \ldots \to C_s \to o) \in \mathcal{F}_m \times \mathcal{M}_m \times \mathcal{T}^\alpha \mid \forall (k,a) \in F.\, M(k) = 0 \wedge M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1\big\}.$$

Elements of $\mathcal{TT}^\alpha_m$ are called *type triples*; they store a type, together with information about flags and markers used while deriving this type. In order to distinguish type triples from types, the former are denoted by letters with a hat, like $\hat\tau$. We also define a function Mk that extracts the marker multiset out of a type triple: $\mathsf{Mk}(\hat\tau) = M$ for $\hat\tau = (F,M,\tau)$. A type triple is *balanced* if $\mathsf{Mk}(\hat\tau) = \mathbf{0}$; otherwise it is *unbalanced*.

Triple containers are used to store type triples that have to be derived for an argument of a lambda-term, or for a lambda-term substituted for a free variable. For balanced type triples, triple containers behave like

sets, that is, they remember only whether every balanced type triple is required or not. Conversely, for unbalanced type triples, triple containers behave like multisets, that is, they remember precisely how many times every unbalanced type triple is required. Thus, formally,

$$\mathcal{TC}^\alpha = \left\{ C \colon \mathcal{TT}^\alpha_{ord(\alpha)} \to \{0, \ldots, |\Sigma|\} \mid \forall \hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)} . \, \mathsf{Mk}(\hat{\tau}) = \mathbf{0} \Rightarrow C(\hat{\tau}) \leq 1 \right\}.$$

For a triple container $C \in \mathcal{TC}^\alpha$ we define $\mathsf{Mk}(C) = \sum_{\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}} \sum_{i=1}^{C(\hat{\tau})} \mathsf{Mk}(\hat{\tau})$. For two triple containers $C, D \in \mathcal{TC}^\alpha$ we define their sum $C \sqcup D \colon \mathcal{TT}^\alpha_{ord(\alpha)} \to \mathbb{N}$ so that for every $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$,

$$(C \sqcup D)(\hat{\tau}) = \begin{cases} C(\hat{\tau}) + D(\hat{\tau}) & \text{if } \mathsf{Mk}(\hat{\tau}) \neq \mathbf{0}\,, \\ \max(C(\hat{\tau}), D(\hat{\tau})) & \text{if } \mathsf{Mk}(\hat{\tau}) = \mathbf{0}\,. \end{cases}$$

We also say that $C \sqsubseteq D$ if $C(\hat{\tau}) = D(\hat{\tau})$ for every unbalanced $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$, and $C(\hat{\tau}) \leq D(\hat{\tau})$ for every balanced $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$. We sometimes write $\{\!|\hat{\tau}_1, \ldots, \hat{\tau}_n|\!\}$ or $\{\!|\hat{\tau}_i \mid i \in \{1, \ldots, n\}|\!\}$ to denote the triple container C such that $C(\hat{\sigma}) = |\{i \in \{1, \ldots, n\} \mid \hat{\tau}_i = \hat{\sigma}\}|$ for every unbalanced type triple $\hat{\sigma}$, and $C(\hat{\sigma}) = 1 \Leftrightarrow \exists i \in \{1, \ldots, n\} \, . \, \hat{\tau}_i = \hat{\sigma}$ for every balanced type triple $\hat{\sigma}$.

A *type environment* is a function Γ that maps every variable x^α to a triple container from $\mathcal{TC}^\alpha$. We use ε to denote the type environment mapping every variable to $\mathbf{0}$. When $\Gamma(x) = \mathbf{0}$, by $\Gamma[x \mapsto C]$ we denote the type environment that maps x to C, and every other variable y to $\Gamma(y)$ (whenever we write $\Gamma[x \mapsto C]$, we implicitly require that $\Gamma(x) = \mathbf{0}$). For two type environments Γ, Γ' we define their sum $\Gamma \sqcup \Gamma'$ so that $(\Gamma \sqcup \Gamma')(x) = \Gamma(x) \sqcup \Gamma'(x)$ for every variable x; moreover, we say that $\Gamma \sqsubseteq \Gamma'$ if $\Gamma(x) \sqsubseteq \Gamma'(x)$ for every variable x.

A *type judgment* is of the form $\Gamma \vdash_m P : \hat{\tau} \triangleright c$, where Γ is a type environment, $m \geq 0$ is called the *order* of the type judgment, P is a lambda-term, $\hat{\tau}$ is an m-bounded type triple of the same sort as P (i.e., $\hat{\tau} \in \mathcal{TT}^\alpha_m$ when P is of sort α), and c is a function $\Sigma \to \mathbb{N}$ called a *flag counter*. Having two functions with values in natural numbers (in particular: two flag counters), $f, g \colon X \to \mathbb{N}$, we write $f \leq g$ when $f(x) \leq g(x)$ for every $x \in X$.

As usually for intersection types, the intuitive meaning of a type $C \to \tau$ is that a lambda-term having this type can return a lambda-term having type τ, while taking an argument for which we can derive all type triples from C. Let us now explain the meaning of a type judgment $\Gamma \vdash_m P : (F, M, \tau) \triangleright c$. Obviously τ is the type derived for P, and Γ contains type triples that could be used for free variables of P in the derivation. As explained above for triple containers, balanced and unbalanced type triples behave differently: all unbalanced type triples assigned to variables by Γ have to be used exactly once in the derivation; conversely, balanced type triples may be used any number of times. Going further, the order m of the type judgment bounds the order of flags and markers that can be used in the derivation: flags can be of order at most $m + 1$, and markers of order at most m. The multiset M counts 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 for free variables in our derivation); we, however, do not include markers provided by arguments of the lambda-term (i.e., coming from the triple containers C_i when $\tau = C_1 \to \ldots \to C_s \to o$). The set F contains information about flags of order at most m used in the derivation. A pair (k, a) can be contained in F if a (k, a)-flag is placed in the derivation itself, or provided by a free variable, or provided by an argument. We do not have to keep in F all such pairs, that is, if we can derive a type triple with some flag set F, then we can derive it also with every subset of F as the flag set. In fact, we cannot keep in F all such pairs due to two restrictions. First, the definition of a flag set

allows to have in F at most one pair (k,a) for every order k. Second, we intentionally remove from F all pairs (k,a) for which $M(k)>0$. Finally, in a type judgment we have a flag counter c, which for each letter a counts the number of $(m+1,a)$-flags present in the derivation.

Type system. Before giving rules of the type system, let us state two general facts. First, all type derivations are assumed to be finite—although we derive types mostly for infinite lambda-terms, each type derivation analyzes only a finite part of a term. Second, we require that premisses and conclusions of all rules are valid type judgments. For example, when the type environment appearing in the conclusion of a rule is $\Gamma \sqcup \Gamma'$, this implies that $\Gamma(x)(\hat\tau)+\Gamma'(x)(\hat\tau)\le|\Sigma|$ holds for all x and all unbalanced type triples $\hat\tau$ (so that $(\Gamma\sqcup\Gamma')(x)$ is indeed a valid triple container). Let us also remark that rules of the type system guarantee that the order m of all type judgments used in a derivation is the same.

Rules of the type system correspond to particular constructs of lambda-calculus. We start by giving the first three rules:

$$\frac{M\restriction_{\le ord(x)} = M'}{\varepsilon[x\mapsto\{\!\{(F,M',\tau)\}\!\}]\vdash_m x:(F,M,\tau)\triangleright\mathbf{0}}\ (\textsc{Var}) \qquad \frac{\Gamma\vdash_m P_i:\hat\tau\triangleright c \qquad i\in\{1,\dots,r\}}{\Gamma\vdash_m \mathsf{nd}\langle P_1,\dots,P_r\rangle:\hat\tau\triangleright c}\ (\textsc{Nd})$$

$$\frac{\Gamma[x\mapsto C']\vdash_m P:(F,M,\tau)\triangleright c \qquad C'\sqsubseteq C}{\Gamma\vdash_m \lambda x.P:(F,M-\mathsf{Mk}(C),C\to\tau)\triangleright c}\ (\lambda)$$

The (VAR) rule allows to have in the resulting marker multiset M some numbers that do not come from the multiset 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 greater than $ord(x)$. This is consistent with the intuitive description of the type system (cf. page 5), which says that a marker of order k can be put in a place that will be a leaf after performing all beta-reductions of order at least k. Indeed, the variable x remains a leaf after performing beta-reductions of orders greater than $ord(x)$, but while performing beta-reductions of order $ord(x)$ this leaf is replaced by a subterm substituted for x. Recall also that, by the definition of a type judgment, we require that $(F,M',\tau)\in\mathcal{TT}^{\alpha}_{ord(x)}$ and $(F,M,\tau)\in\mathcal{TT}^{\alpha}_{m}$, for an appropriate sort α; this introduces a bound on maximal numbers that may appear in F and M.

Example 3.2. In this example, as well as in the next three examples, we illustrate particular rules of the type system while deriving type judgments for subterms of the lambda-term $\Lambda(\mathcal{G}_1)$ from Example 2.1. Denoting $\hat\rho_0^{all}=(\emptyset,\{\!\{0\}\!\},o)$ we can derive:

$$\frac{}{\varepsilon[\mathsf{x}\mapsto\{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{x}:(\emptyset,\{\!\{0\}\!\},o)\triangleright\mathbf{0}}\ (\textsc{Var}) \qquad \frac{}{\varepsilon[\mathsf{x}\mapsto\{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{x}:(\emptyset,\{\!\{0,1,1,1\}\!\},o)\triangleright\mathbf{0}}\ (\textsc{Var})$$

In the second derivation, three markers of order 1 are placed in the conclusion of the rule. We could equally well place one or two such markers (but not four, because in our examples we assume that $|\Sigma|=3$). □

We see that in order to derive a type for the nondeterministic choice $\mathsf{nd}\langle P_1,\dots,P_r\rangle$, we need to derive it for one of the subterms $P_1,\dots,P_r$.

For the (λ) rule, recall that $C'\sqsubseteq C$ means that in C' we have all unbalanced type triples from C, and some subset of balanced type triples from C. Thus in a subderivation concerning the lambda-term P, we need to use all unbalanced type triples provided by an argument of $\lambda x.P$, while balanced type triples may be used or not. Recall also that we intend to store in the marker multiset information about markers contained in the derivation itself and those provided by free variables, but not those provided by arguments.

Because of this, in the conclusion of the rule we remove from M information about markers provided by x. It is required, implicitly, that the result remains nonnegative. The set F, unlike M, stores also flags provided by arguments, so we do not need to remove anything from F.

Example 3.3. In this example we show how the (ND) and (λ) rules can be used. The derivations work for every flag counter c. Notice that in the conclusion of the (λ) rule, in both derivations, we remove 0 from the marker multiset, because an order-0 marker is provided by x.

$$\dfrac{\dfrac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright c}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright c}\ (\text{ND})}{\varepsilon \vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!|\hat{\rho}_0^{all}|\!\} \to o) \triangleright c}\ (\lambda)$$

$$\dfrac{\dfrac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset, \{\!|0,1,1,1|\!\}, o) \triangleright c}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset, \{\!|0,1,1,1|\!\}, o) \triangleright c}\ (\text{ND})}{\varepsilon \vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset, \{\!|1,1,1|\!\}, \{\!|\hat{\rho}_0^{all}|\!\} \to o) \triangleright c}\ (\lambda)$$

□

The next three rules use a predicate $Comp_m$, saying how flags and markers from premisses contribute to the conclusion. It takes "as input" pairs (F_i, c_i) for $i \in I$, consisting of a flag set F_i and a flag counter c_i from some premiss. Moreover, the predicate takes a marker multiset M that appears in the conclusion of the rule. The goal is to compute a flag set F and a flag counter c that should be placed in the conclusion. First, for each $k \in \{2, \ldots, m+1\}$ consecutively, we decide which flags of order k should be placed in the considered node of a type derivation (the $Comp_m$ predicate is not responsible for placing flags of order 1). We follow here the rules mentioned in the intuitive description. Namely, we place a (k,a)-flag if we are on the path leading to a marker of order $k-1$ (i.e., if $M(k-1) > 0$), and simultaneously we receive information about a $(k-1,a)$-flag. By receiving the information we mean that either a $(k-1,a)$-flag was placed in the current node, or $(k-1,a)$ belongs to some set F_i. Actually, we place multiple (k,a)-flags: one per each $(k-1,a)$-flag placed in the current node, and one per each set F_i containing $(k-1,a)$. Then, we compute F and c. In $c(a)$, for every $a \in \Sigma$, we store the number of $(m+1,a)$-flags: we sum all the flag counters c_i, and we add the number of $(m+1,a)$-flags placed in the current node. In F, we allow to keep elements of all F_i, and we allow to add pairs (k,a) for flags that were placed in the current node, but it can be chosen "nondeterministically" which of them are actually taken to F, and which are dropped. It is often necessary to drop some elements, since when the set F is used in a type triple, the definitions of a flag set and of a type triple put additional requirements on this set.

Below we give a formal definition, in which $f'_{k,a}$ contains the number of (k,a)-flags placed in the current node, while $f_{k,a}$ additionally counts the number of premisses for which $(k,a) \in F_i$. We say that $(F,c) \in Comp_m(M; ((F_i,c_i))_{i\in I})$ when

$$F \subseteq \{(k,a) \mid f_{k,a} > 0\}, \qquad \text{and} \qquad c(a) = f_{m+1,a} + \sum_{i\in I} c_i(a) \qquad \text{for all } a \in \Sigma,$$

where, for $k \in \{1, \ldots, m+1\}$ and $a \in \Sigma$,

$$f_{k,a} = f'_{k,a} + \sum_{i\in I} |F_i \cap \{(k,a)\}|, \qquad f'_{k,a} = \begin{cases} 0 & \text{if } k = 1 \text{ or } M(k-1) = 0, \\ f_{k-1,a} & \text{otherwise.} \end{cases}$$

We now present rules for node constructors using letters other than nd (recall that the type system is intended to work only for word-recognizing schemes, so it is enough to handle node constructors of arity at most 1):

$$\frac{(F,c) \in \mathit{Comp}_m(M;(\{(1,a)\},\mathbf{0})) \qquad a \neq \mathsf{nd}}{\varepsilon \vdash_m a\langle\rangle : (F,M,o) \triangleright c} \text{ (CON0)}$$

$$\frac{\Gamma \vdash_m P : (F_1,M,o) \triangleright c_1 \qquad (F,c) \in \mathit{Comp}_m(M;(\{(1,a)\},\mathbf{0}),(F_1,c_1)) \qquad a \neq \mathsf{nd}}{\Gamma \vdash_m a\langle P\rangle : (F,M,o) \triangleright c} \text{ (CON1)}$$

By passing the set $\{(1,a)\}$ to Comp_m we express the fact that a $(1,a)$-flag is placed in the current node. We remark that the set $\{(1,a)\}$, passed to Comp_0, is not an element of $\mathcal{F}_0 = \{\emptyset\}$ (and the definition of Comp_m does not require this). In the (CON0) rule, that is, if we are in a leaf, we are allowed to place markers of an arbitrary order: the marker multiset M may be arbitrary.

Remark. We allow to put markers only in leaves (i.e., in the (VAR) and (CON0) rules). Equally well it could be allowed to put markers in any node of a derivation; this does not change a lot. Actually, Asada and Kobayashi (2017), while using our type system to develop a pumping lemma, allow to place markers anywhere.

Example 3.4. For $a \in \{\mathsf{a},\mathsf{b},\mathsf{c}\}$, let c_a be the flag counter such that $c_a(a) = 1$ and $c_a(b) = 0$ for all $b \in \Sigma \setminus \{a\}$. The (CON1) rule may be instantiated in the following ways:

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{x} : (\emptyset,\{\!|0|\!\},o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\},\{\!|0|\!\},o) \triangleright \mathbf{0}} \text{ (CON1)}$$

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{x} : (\emptyset,\{\!|0,1,1,1|\!\},o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset,\{\!|0,1,1,1|\!\},o) \triangleright c_{\mathsf{a}}} \text{ (CON1)}$$

In the first example, a $(1,\mathsf{a})$-flag is placed in the conclusion of the rule. Indeed, the (CON1) rule implies that the pair $(1,\mathsf{a})$ is passed to the Comp_1 predicate. In the second derivation, additionally a $(2,\mathsf{a})$-flag is placed in the conclusion of the rule: since the marker multiset contains 1 (order-1 markers are visible), we do not put $(1,\mathsf{a})$ to the flag set, but instead we create a $(2,\mathsf{a})$-flag, which results in increasing the flag counter. □

The last rule describes application:

$$\frac{\begin{array}{c}\Gamma' \vdash_m P : (F',M',\{\!|(F_i{\restriction}_{\leq ord(Q)},M_i{\restriction}_{\leq ord(Q)},\tau_i) \mid i \in I|\!\} \to \tau) \triangleright c' \\ \Gamma_i \vdash_m Q : (F_i,M_i,\tau_i) \triangleright c_i \text{ for each } i \in I \qquad M = M' + \sum_{i\in I} M_i \qquad \mathit{ord}(Q) \leq m \\ (F,c) \in \mathit{Comp}_m(M;(F',c'),((F_i{\restriction}_{>ord(Q)},c_i))_{i\in I}) \qquad \{(k,a) \in F' \mid M(k) = 0\} \subseteq F\end{array}}{\Gamma' \sqcup \bigsqcup_{i\in I} \Gamma_i \vdash_m P\,Q : (F,M,\tau) \triangleright c} \text{ (@)}$$

In this rule, it is allowed (and potentially useful) that for two different $i \in I$ the type triples (F_i,M_i,τ_i) are equal. It is also allowed that $I = \emptyset$, in which case no type needs to be derived for Q. Observe how flags and markers coming from premisses concerning Q are propagated: only flags and markers of orders

$k \leq ord(Q)$ are visible to P, while only flags of orders $k > ord(Q)$ are passed to the $Comp_m$ predicate. This can be justified if we recall the intuitions staying behind the type system (cf. page 5). Indeed, while considering flags and markers of order k, we should imagine the lambda-term obtained from the current lambda-term by performing all beta-reductions of order at least k; the distribution of flags and markers of order k in the current lambda-term actually simulates their distribution in this imaginary lambda-term. Thus, if $ord(Q) \geq k$, then our application will disappear in this imaginary lambda-term, and Q will be already substituted somewhere in P; for this reason we need to pass information about flags and markers of order k from Q to P. Conversely, if $ord(Q) < k$, then in the imaginary lambda-term the considered application will be still present, and, in consequence, the subterm corresponding to P will not see flags and markers of order k placed in the subterm corresponding to Q. The condition $\{(k,a) \in F' \mid M(k) = 0\} \subseteq F$ (saying that flags derived for P cannot disappear, unless they meet information about markers of the corresponding order) is added for technical reasons. It turns out to be useful in our proofs, namely, in the proof of Lemma 7.1 on page 39.

Example 3.5. Recalling that $\hat{\rho}_0^{all} = (\emptyset, \{\!|0|\!\}, o)$, denote by $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ the type triples derived in Example 3.3: $\hat{\tau}_{\mathsf{a}} = (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)$ and $\hat{\tau}_{\mathsf{m}} = (\emptyset, \{\!|1,1,1|\!\}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)$. We can derive:

$$\dfrac{\dfrac{\overline{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_{\mathsf{a}}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}} \qquad \dfrac{\overline{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_{\mathsf{m}}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_{\mathsf{m}} \triangleright \mathbf{0}} \qquad \overline{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_{\mathsf{m}}|\!\}, \mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{f}\,\mathsf{y} : (\emptyset, \{\!|0,1,1,1|\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{m}}|\!\}, \mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : (\emptyset, \{\!|0,1,1,1|\!\}, o) \triangleright c_{\mathsf{a}}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{m}}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_{\mathsf{m}} \triangleright c_{\mathsf{a}}}\,(\lambda)$$

Below the lower (@) rule information about a $(1,\mathsf{a})$-flag (from the first premiss) meets information about a marker of order 1 (from the second premiss), and thus a $(2,\mathsf{a})$-flag is placed, which increases the flag counter. □

Denote $\hat{\rho}_m^{all} = (\emptyset, M_m^{all}, o)$, where $M_m^{all} \in \mathcal{M}_m$ is such that $M_m^{all}(0) = 1$ and $M_m^{all}(k) = |\Sigma|$ for all $k \in \{1,\ldots,m\}$. The key property of the type system is described by the following theorem.

Theorem 3.2. *Let $m \in \mathbb{N}$, let P be a word-recognizing closed lambda-term of sort o and of complexity at most $m+1$, and let $A \subseteq \Sigma$. Then $SUP_A(\mathcal{L}(BT(P)))$ holds if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$.*

We postpone the proof of this theorem to Sections 4-7. Before this proof, we give some examples and comments.

Example 3.6. In Examples 3.6-3.9 we present a complete derivation concerning the recursion scheme $\mathcal{G}_1$ from Example 2.1, continuing Examples 3.2-3.5. Actually, we present a series of derivations with larger and larger values in the flag counter; this corresponds to the fact that the language of $\mathcal{G}_1$ contains words with arbitrarily many occurrences of the letters a and b. The four examples follow the intuitions shown in Example 3.1.

We start by a derivation of order 0 that works for $BT(\Lambda(\mathcal{G}_1))$. Recall that $BT(\Lambda(\mathcal{G}_1)) = \mathsf{nd}\langle T_{2^0}, \mathsf{nd}\langle T_{2^1}, \mathsf{nd}\langle T_{2^2}, \ldots\rangle\rangle\rangle$, where $T_0 = \mathsf{c}\langle\rangle$ and $T_{i+1} = \mathsf{nd}\langle \mathsf{a}\langle T_i\rangle, \mathsf{b}\langle T_i\rangle\rangle$.

By applying the (CON0) rule we can derive $\varepsilon \vdash_0 \mathsf{c}\langle\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{c}}$ (recall that $\hat{\rho}_0^{all} = (\emptyset, \{\!|0|\!\}, o)$. Then,

having a derivation of $\varepsilon \vdash_0 T_i : \hat{\rho}_0^{all} \triangleright c$, we can continue in one of two ways:

$$\dfrac{\dfrac{\varepsilon \vdash_0 T_i : \hat{\rho}_0^{all} \triangleright c}{\varepsilon \vdash_0 \mathsf{a}\langle T_i\rangle : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{a}}}\,(\textsc{Con}1)}{\varepsilon \vdash_0 T_{i+1} : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{a}}}\,(\textsc{Nd}) \qquad \dfrac{\dfrac{\varepsilon \vdash_0 T_i : \hat{\rho}_0^{all} \triangleright c}{\varepsilon \vdash_0 \mathsf{b}\langle T_i\rangle : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{b}}}\,(\textsc{Con}1)}{\varepsilon \vdash_0 T_{i+1} : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{b}}}\,(\textsc{Nd})$$

In consequence, we can derive $\varepsilon \vdash_0 T_i : \hat{\rho}_0^{all} \triangleright c$ for every flag counter c such that $c(\mathsf{a}) + c(\mathsf{b}) = i$ and $c(\mathsf{c}) = 1$; this reflects precisely the number of occurrences of particular letters in words from $\mathcal{L}(T_i)$. For every i of the form 2^j we can continue by deriving $\varepsilon \vdash_0 BT(\Lambda(\mathcal{G}_1)) : \hat{\rho}_0^{all} \triangleright c$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varepsilon \vdash_0 T_{2^j} : \hat{\rho}_0^{all} \triangleright c}{\varepsilon \vdash_0 \mathsf{nd}\langle T_{2^j}, \ldots\rangle : \hat{\rho}_0^{all} \triangleright c}\,(\textsc{Nd})}{\cdots}\,(\textsc{Nd})}{\varepsilon \vdash_0 \mathsf{nd}\langle T_{2^1}, \ldots \mathsf{nd}\langle T_{2^j}, \ldots\rangle \ldots\rangle : \hat{\rho}_0^{all} \triangleright c}\,(\textsc{Nd})}{\varepsilon \vdash_0 \mathsf{nd}\langle T_{2^0}, \mathsf{nd}\langle T_{2^1}, \ldots \mathsf{nd}\langle T_{2^j}, \ldots\rangle \ldots\rangle\rangle : \hat{\rho}_0^{all} \triangleright c}\,(\textsc{Nd})$$

□

Example 3.7. Next, we consider the lambda-term obtained from $\Lambda(\mathcal{G}_1)$ by performing all (infinitely many) beta-reductions of order 1. It is the lambda-term Q_0 where for all $i \in \mathbb{N}$,

$$Q_i = \mathsf{nd}\langle S_i\,(\mathsf{c}\langle\rangle), Q_{i+1}\rangle\,, \qquad S_{i+1} = \lambda \mathsf{y}.S_i\,(S_i\,\mathsf{y})\,, \qquad \text{and} \qquad S_0 = \lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle\,.$$

Denote $\hat{\tau}_0 = (\emptyset, \mathbf{0}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)$. We can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{x} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{a}\langle \mathsf{x}\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{a}}}\,(\textsc{Con}1)}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{a}}}\,(\textsc{Nd})}{\varepsilon \vdash_0 S_0 : \hat{\tau}_0 \triangleright c_{\mathsf{a}}}\,(\lambda) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{x} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{b}\langle \mathsf{x}\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{b}}}\,(\textsc{Con}1)}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{b}}}\,(\textsc{Nd})}{\varepsilon \vdash_0 S_0 : \hat{\tau}_0 \triangleright c_{\mathsf{b}}}\,(\lambda)$$

Then, having derivations for S_i with flag counters c_1, c_2, we can continue as follows:

$$\dfrac{\dfrac{\varepsilon \vdash_0 S_i : \hat{\tau}_0 \triangleright c_1 \qquad \dfrac{\varepsilon \vdash_0 S_i : \hat{\tau}_0 \triangleright c_2 \qquad \dfrac{}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 S_i\,\mathsf{y} : \hat{\rho}_0^{all} \triangleright c_2}\,(@)}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_0 S_i\,(S_i\,\mathsf{y}) : \hat{\rho}_0^{all} \triangleright c_1 + c_2}\,(@)}{\varepsilon \vdash_0 S_{i+1} : \hat{\tau}_0 \triangleright c_1 + c_2}\,(\lambda)$$

In consequence, we can derive $\varepsilon \vdash_0 S_i : \hat{\tau}_0 \triangleright c$ for every flag counter c such that $c(\mathsf{a}) + c(\mathsf{b}) = 2^i$ and $c(\mathsf{c}) = 0$. We finish the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varepsilon \vdash_0 S_i : \hat{\tau}_0 \triangleright c \qquad \dfrac{}{\varepsilon \vdash_0 \mathsf{c}\langle\rangle : \hat{\rho}_0^{all} \triangleright c_{\mathsf{c}}}\,(\textsc{Con}0)}{\varepsilon \vdash_0 S_i\,(\mathsf{c}\langle\rangle) : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{c}}}\,(@)}{\varepsilon \vdash_0 Q_i : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{c}}}\,(\textsc{Nd})}{\varepsilon \vdash_0 Q_{i-1} : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{c}}}\,(\textsc{Nd})}{\cdots}\,(\textsc{Nd})}{\varepsilon \vdash_0 Q_0 : \hat{\rho}_0^{all} \triangleright c + c_{\mathsf{c}}}\,(\textsc{Nd})$$

As in the previous example, to every derivation we can assign a corresponding word in $\mathcal{L}(BT(Q_0))$, so that the flag counter computes precisely the number of occurrences of particular letters in the word. □

Example 3.8. Let us now increase the order of the derivation from the previous example, by adding markers of order 1 and flags of order 2. Recall the type triples defined previously:

$$\hat{\rho}_0^{all} = (\emptyset, \{\!|0|\!\}, o)\,, \qquad \hat{\tau}_{\mathsf{a}} = (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)\,,$$
$$\hat{\rho}_1^{all} = (\emptyset, \{\!|0,1,1,1|\!\}, o)\,, \qquad \hat{\tau}_{\mathsf{m}} = (\emptyset, \{\!|1,1,1|\!\}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)\,.$$

From Examples 3.2-3.4 we already know how to derive $S_0 : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}$ and $S_0 : \hat{\tau}_{\mathsf{m}} \triangleright c_{\mathsf{a}}$. Then, consecutively for every $i \in \mathbb{N}$, based on a type judgment for S_i, we derive a type judgment for S_{i+1}:

$$\dfrac{\dfrac{\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0} \qquad \dfrac{\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0} \qquad \overline{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 S_i\,\mathsf{y} : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 S_i\,(S_i\,\mathsf{y}) : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon \vdash_1 \lambda\mathsf{y}.S_i\,(S_i\,\mathsf{y}) : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}}\,(\lambda)$$

Next, suppose that we can derive $\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}$ and $\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{m}} \triangleright c$. Out of those type judgments we can derive (notice a high similarity to the derivation from Example 3.5):

$$\dfrac{\dfrac{\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0} \qquad \dfrac{\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{m}} \triangleright c \qquad \overline{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 S_i\,\mathsf{y} : \hat{\rho}_1^{all} \triangleright c}\,(@)}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 S_i\,(S_i\,\mathsf{y}) : \hat{\rho}_1^{all} \triangleright c + c_{\mathsf{a}}}\,(@)}{\varepsilon \vdash_1 \lambda\mathsf{y}.S_i\,(S_i\,\mathsf{y}) : \hat{\tau}_{\mathsf{m}} \triangleright c + c_{\mathsf{a}}}\,(\lambda)$$

In analogy to $\hat{\tau}_{\mathsf{a}}$, let $\hat{\tau}_{\mathsf{b}} = (\{(1,\mathsf{b})\}, \mathbf{0}, \{\!|\hat{\rho}_0^{all}|\!\} \to o)$. Changing a to b in the above derivations, we can derive a type judgment $\varepsilon \vdash_1 S_0 : \hat{\tau}_{\mathsf{b}} \triangleright \mathbf{0}$, then $\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{b}} \triangleright \mathbf{0}$ for every $i \in \mathbb{N}$, and then out of $\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{m}} \triangleright c$ we can derive $\varepsilon \vdash_1 S_{i+1} : \hat{\tau}_{\mathsf{m}} \triangleright c + c_{\mathsf{b}}$. In consequence, for every $i \in \mathbb{N}$ we can derive $\varepsilon \vdash_1 S_i : \hat{\tau}_{\mathsf{m}} \triangleright c$ for any flag counter c such that $c(\mathsf{a}) + c(\mathsf{b}) = i + 1$ and $c(\mathsf{c}) = 0$. Recall that words described by S_i contain 2^i letters, so this time values in the flag counter are exponentially smaller (unlike for $\vdash_0$ in the previous example).

As previously, it is easy to finish the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varepsilon \vdash_0 S_i : \hat{\tau}_{\mathsf{m}} \triangleright c \qquad \overline{\varepsilon \vdash_0 \mathsf{c}\langle\rangle : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\,(\textsc{Con}0)}{\varepsilon \vdash_0 S_i\,(\mathsf{c}\langle\rangle) : \hat{\rho}_1^{all} \triangleright c}\,(@)}{\varepsilon \vdash_0 Q_i : \hat{\rho}_1^{all} \triangleright c}\,(\textsc{Nd})}{\varepsilon \vdash_0 Q_{i-1} : \hat{\rho}_1^{all} \triangleright c}\,(\textsc{Nd})}{\cdots}\,(\textsc{Nd})}{\varepsilon \vdash_0 Q_0 : \hat{\rho}_1^{all} \triangleright c}\,(\textsc{Nd})$$

Notice that in the (Con0) rule a $(1,\mathsf{c})$-flag is placed. It is not necessary to remember information about this flag in the type triple, and thus we can derive $\hat{\rho}_0^{all}$, having an empty flag set. □

Example 3.9. Finally, we actually present a derivation for the lambda-term $\Lambda(\mathcal{G}_1)$. Besides the type triples $\hat{\rho}_0^{all}, \hat{\rho}_1^{all}, \hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{b}, \hat{\tau}_\mathsf{m}$ defined previously let us also define

$$\hat{\sigma}_R = (\emptyset, \{\!|0|\!\}, \{\!|\hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{b}, \hat{\tau}_\mathsf{m}|\!\} \to o)\,.$$

Recall from Example 2.1 that R_1 is a lambda-term such that $R_1 = \lambda\mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. A type judgment $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \mathbf{0}$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_\mathsf{m} \triangleright \mathbf{0}}\,(\textsc{Var}) \qquad \dfrac{}{\varepsilon \vdash_1 \mathsf{c}\langle\rangle : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\,(\textsc{Con0})}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \mathsf{f}\,(\mathsf{c}\langle\rangle) : \hat{\rho}_1^{all} \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle : \hat{\rho}_1^{all} \triangleright \mathbf{0}}\,(\textsc{Nd})}{\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \mathbf{0}}\,(\lambda)$$

Notice that the type triples $\hat{\tau}_\mathsf{a}$ and $\hat{\tau}_\mathsf{b}$ required for the argument by $\hat{\sigma}_R$ are not used here; recall that the (λ) rule allows to discard them, since they are balanced. On the other hand, the type triple $\hat{\tau}_\mathsf{m}$ is unbalanced, so it could not be discarded, and has to be used exactly once in the derivation.

Next, we derive the same type triple for R_1, but using the second argument of the nd symbol; this results in greater values of the flag counter. In Example 3.5 we have derived the type judgment $\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{m} \triangleright c_\mathsf{a}$. Similarly we can derive $\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{b}, \hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{m} \triangleright c_\mathsf{b}$. We continue by deriving the type triple $\hat{\tau}_\mathsf{a}$ for the subterm $\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0}} \qquad \dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0}} \qquad \dfrac{}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}, \mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{f}\,\mathsf{y} : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}, \mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0}}\,(\lambda)$$

In the above derivation there are no flags nor markers. Similarly we can derive $\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{b}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{b} \triangleright \mathbf{0}$. We continue with the lambda-term R_1 (for an arbitrary flag counter c):

$$\dfrac{\dfrac{\dfrac{\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c \qquad \varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0}}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{b}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{b} \triangleright \mathbf{0}} \qquad \dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_\mathsf{m} \triangleright c_\mathsf{a}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{b}, \hat{\tau}_\mathsf{m}|\!\}] \vdash_1 R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y})) : \hat{\rho}_1^{all} \triangleright c + c_\mathsf{a}}\,(\textsc{Nd})}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}, \hat{\tau}_\mathsf{b}, \hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle : \hat{\rho}_1^{all} \triangleright c + c_\mathsf{a}}\,(\lambda)$$
$$\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c + c_\mathsf{a}$$

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 and third premisses provide $(1,\mathsf{a})$- and $(1,\mathsf{b})$-flags while the last premiss provides markers of order 1. We recall from the definition of the (@) rule that information about flags and markers coming from the arguments is divided into two parts. Numbers not greater than the order of the argument (which is 1 in our case) are passed to the operator, while only greater numbers (in our case: greater than 1) contribute in creating new flags via the $Comp$ predicate.

Similarly, out of $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c + c_\mathsf{b}$. By composing these two derivation fragments, we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ for every c such that $c(\mathsf{c}) = 0$ (by modifying

slightly the initial fragment of the derivation descending to $\mathsf{f}\,(\mathsf{c}\langle\rangle)$ we could also obtain $c(\mathsf{c}) = 1$, but not $c(\mathsf{c}) > 1$). Finally, we apply the argument $S_0 = \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle$, and we derive for $\Lambda(\mathcal{G}_1)$ the type triple $\hat{\rho}_1^{all}$, appearing in Theorem 3.2.

$$\frac{\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c \qquad \varepsilon \vdash_1 S_0 : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0} \qquad \varepsilon \vdash_1 S_0 : \hat{\tau}_\mathsf{b} \triangleright \mathbf{0} \qquad \varepsilon \vdash_1 S_0 : \hat{\tau}_\mathsf{m} \triangleright c_\mathsf{a}}{\varepsilon \vdash_1 \Lambda(\mathcal{G}_1) : \hat{\rho}_1^{all} \triangleright c + c_\mathsf{a}}\ (@)$$

Recall that from Examples 3.2-3.4 we already know how to derive the three premisses concerning S_0. There is a lack of symmetry here with respect to letters a and b, but instead of the last premiss we could equally well use $\varepsilon \vdash_1 S_0 : \hat{\tau}_\mathsf{m} \triangleright c_\mathsf{b}$, obtaining flag counter $c + c_\mathsf{b}$ at the end. We notice that there is no direct correspondence between the considered derivation and some particular word in $\mathcal{L}(\mathcal{G}_1)$; we can only say that such a derivation with flag counter $c + c_\mathsf{a}$ talks about words from $\mathcal{L}(\mathcal{G}_1)$ having $2^{c(\mathsf{a})+c(\mathsf{b})} + 1$ nodes, because $c(\mathsf{a}) + c(\mathsf{b})$ times we have descended to the second argument of the nd symbol in R_1. We remark that in every of the above derivations only four flags of order 1 are present (two $(1, \mathsf{a})$-flags, one $(1, \mathsf{b})$-flag, and one $(1, \mathsf{c})$-flag), in the four nodes using rules (CON1) or (CON0). □

Example 3.10. This is a "negative" example: we have here a scheme with a finite language. Due to Theorem 3.2, type derivations concerning such a scheme result in small values in the flag counter. Namely, consider a scheme $\mathcal{G}_2$, similar to $\mathcal{G}_1$, where

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle)\,, \qquad\qquad \mathcal{R}(\mathsf{N}) = \lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), \mathsf{N}\,(\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y})\rangle\,.$$

The only difference, compared with $\mathcal{G}_1$, is that in $\mathcal{R}(\mathsf{N})$ we have replaced $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$ by $\mathsf{f}\,\mathsf{y}$. In consequence, the lambda-term R_2 (defined analogously to R_1 from Example 2.1) is obtained from R_1 by replacing all occurrences of the subterm $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$ with $\mathsf{f}\,\mathsf{y}$. We have $BT(\Lambda(\mathcal{G}_2)) = \mathsf{nd}\langle T_1, \mathsf{nd}\langle T_1, \mathsf{nd}\langle T_1, \ldots\rangle\rangle\rangle$, where $T_1 = \mathsf{nd}\langle\mathsf{a}\langle\mathsf{c}\langle\rangle\rangle, \mathsf{b}\langle\mathsf{c}\langle\rangle\rangle\rangle$, and thus $\mathcal{L}(\mathcal{G}_2) = \{\mathsf{a}\langle\mathsf{c}\langle\rangle\rangle, \mathsf{b}\langle\mathsf{c}\langle\rangle\rangle\}$.

Let us see how the type derivations have to be changed. The type judgment $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright \mathbf{0}$ can be obtained without any change, as its derivation descends to the first child of the outermost $\mathsf{nd}\langle\cdot,\cdot\rangle$ in $R_2 = \lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), R_2\,(\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y})\rangle$. The type judgment $\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{a}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,\mathsf{y} : \hat{\tau}_\mathsf{a} \triangleright \mathbf{0}$, and a similar one for $\hat{\tau}_\mathsf{b}$, can be obtained without any problem, as the type judgments concerning the subterms $\mathsf{f}\,\mathsf{y}$ and $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$ were the same. Let us now see what happens to the derivation of the type triple $\hat{\tau}_\mathsf{m}$:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_\mathsf{m} \triangleright \mathbf{0}}\ (\textsc{Var}) \qquad \dfrac{}{\varepsilon[\mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\ (\textsc{Var})}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}, \mathsf{y} \mapsto \{\!|\hat{\rho}_0^{all}|\!\}] \vdash_1 \mathsf{f}\,\mathsf{y} : (\emptyset, \{\!|0, 1, 1, 1|\!\}, o) \triangleright \mathbf{0}}\ (@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{m}|\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,\mathsf{y} : \hat{\tau}_\mathsf{m} \triangleright \mathbf{0}}\ (\lambda)$$

In the subterm $\mathsf{f}\,\mathsf{y}$ we have only one occurrence of f, so we cannot use simultaneously both $\hat{\tau}_\mathsf{a}$ and $\hat{\tau}_\mathsf{m}$ (as we did for $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$); as a result, no order-2 flag is placed. Thus if we create a derivation for R_2 that descends to the second child of the outermost $\mathsf{nd}\langle\cdot,\cdot\rangle$, out of $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright \mathbf{0}$ we derive again $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright \mathbf{0}$, without any change to the flag counter. In consequence, the type triple $\hat{\rho}_1^{all}$ is derived for $\Lambda(\mathcal{G}_2)$ with flag counter c_a (or c_b, if one prefers). This corresponds to the fact that $\mathcal{L}(\mathcal{G}_2)$ contains only words with one letter among a, b. □

Example 3.11. This example shows how subterms disappearing during beta-reductions are handled by our type system. Consider a lambda-term $P_2 = (\lambda\mathsf{g}.\Lambda(\mathcal{G}_2))\,(\lambda\mathsf{z}.P_\mathsf{a})$, where P_a is the unique lambda-term

such that $P_{\mathsf{a}} = \mathsf{nd}\langle \mathsf{z}, \mathsf{a}\langle P_{\mathsf{a}}\rangle\rangle$ (by the way, notice that P_2 is a lambda-term that cannot be described by any scheme, because the free variable z occurs in P_{a} infinitely many times). We see that P_2 beta-reduces to $\Lambda(\mathcal{G}_2)$, hence the recognized language remains unchanged. Let us see what happens on the side of type derivations. Notice that we can create the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{z}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{z}:\hat\rho_0^{all}\triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{z}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 P_{\mathsf{a}}:\hat\rho_0^{all}\triangleright \mathbf{0}}\,(\textsc{Nd})}{\varepsilon[\mathsf{z}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{a}\langle P_{\mathsf{a}}\rangle:(\{(1,\mathsf{a})\},\{\!\{0\}\!\},o)\triangleright \mathbf{0}}\,(\textsc{Con1})}{\varepsilon[\mathsf{z}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 P_{\mathsf{a}}:(\{(1,\mathsf{a})\},\{\!\{0\}\!\},o)\triangleright \mathbf{0}}\,(\textsc{Nd})}{\varepsilon\vdash_1 \lambda\mathsf{z}.P_{\mathsf{a}}:\hat\tau_{\mathsf{a}}\triangleright \mathbf{0}}\,(\lambda)$$

The two lines above the (λ) rule can be repeated arbitrarily many times. Then, in the conclusion of every (CON1) rule, a $(1,\mathsf{a})$-flag is placed (there are no $(2,\mathsf{a})$-flags, though). Such a derivation can be used as a part of a derivation for P_2:

$$\dfrac{\dfrac{\varepsilon\vdash_1 \Lambda(\mathcal{G}_2):\hat\rho_1^{all}\triangleright c_{\mathsf{a}}}{\varepsilon\vdash_1 \lambda\mathsf{g}.\Lambda(\mathcal{G}_2):(\emptyset,\{\!\{0,1,1,1\}\!\},\{\!\{\hat\tau_{\mathsf{a}}\}\!\}\to o)\triangleright c_{\mathsf{a}}}\,(\lambda) \qquad \varepsilon\vdash_1 \lambda\mathsf{z}.P_{\mathsf{a}}:\hat\tau_{\mathsf{a}}\triangleright \mathbf{0}}{\varepsilon\vdash_1 P_2:\hat\rho_1^{all}\triangleright c_{\mathsf{a}}}\,(@)$$

Because $\hat\tau_{\mathsf{a}}$ is balanced, it can be discarded in the (λ) rule; it does not need to be used in the derivation for $\Lambda(\mathcal{G}_2)$. We thus obtain a derivation for P_2 in which there are many $(1,\mathsf{a})$-flags (but only one $(2,\mathsf{a})$-flag). 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). □

Example 3.12. In the derivation from Example 3.9 all order-1 markers were placed in the same leaf, corresponding to the subterm x. In this example we see that sometimes it is necessary to place markers in multiple leaves. To this end, consider a scheme $\mathcal{G}_3$, where additionally to M and N we have a nonterminal $\mathsf{M_b}$ of sort o, and the rules are changed to:

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\lambda\mathsf{x}.\mathsf{a}\langle\mathsf{x}\rangle)\,, \qquad \mathcal{R}(\mathsf{M_b}) = \mathsf{nd}\langle\mathsf{c}\langle\rangle, \mathsf{b}\langle\mathsf{M_b}\rangle\rangle\,,$$
$$\mathcal{R}(\mathsf{N}) = \lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,\mathsf{M_b}, \mathsf{N}\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle\,.$$

Here we need to place one order-1 marker (responsible for counting occurrences of the a letter) in a leaf corresponding to x, and another order-1 marker (responsible for counting occurrences of the b letter) in a leaf corresponding to $\mathsf{c}\langle\rangle$. We do not have to care where to put the third available order-1 marker; let us put it in a leaf corresponding to x.

Since we want to place only two order-1 markers in a leaf corresponding to x, this time we consider the type triple $\hat\tau'_{\mathsf{m}} = (\emptyset,\{\!\{1,1\}\!\},\{\!\{\hat\rho_0^{all}\}\!\}\to o)$. We can derive $\hat\tau'_{\mathsf{m}}$ and $\hat\tau_{\mathsf{a}} = (\{(1,\mathsf{a})\},\mathbf{0},\{\!\{\hat\rho_0^{all}\}\!\}\to o)$ for $\lambda\mathsf{x}.\mathsf{a}\langle\mathsf{x}\rangle$:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{x}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{x}:(\emptyset,\{\!\{0,1,1\}\!\},o)\triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x}\mapsto \{\!\{\hat\rho_0^{all}\}\!\}]\vdash_1 \mathsf{a}\langle\mathsf{x}\rangle:(\emptyset,\{\!\{0,1,1\}\!\},o)\triangleright c_{\mathsf{a}}}\,(\textsc{Con1})}{\varepsilon\vdash_1 \lambda\mathsf{x}.\mathsf{a}\langle\mathsf{x}\rangle:\hat\tau'_{\mathsf{m}}\triangleright c_{\mathsf{a}}}\,(\lambda)$$

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0^{all}\}\!\}] \vdash_1 \mathsf{x} : \hat{\rho}_0^{all} \triangleright \mathbf{0}}\ (\textsc{Var})}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0^{all}\}\!\}] \vdash_1 \mathsf{a}\langle \mathsf{x}\rangle : (\{(1,\mathsf{a})\}, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}}\ (\textsc{Con}1)}{\varepsilon \vdash_1 \lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}}\ (\lambda)$$

Denote by P_{b} the lambda-term corresponding to M_{b}, that is, let $P_{\mathsf{b}} = \mathsf{nd}\langle \mathsf{c}\langle\rangle, \mathsf{b}\langle P_{\mathsf{b}}\rangle\rangle$. We can derive:

$$\dfrac{\dfrac{}{\varepsilon \vdash_1 \mathsf{c}\langle\rangle : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c_{\mathsf{c}}}\ (\textsc{Con}0)}{\varepsilon \vdash_1 P_{\mathsf{b}} : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c_{\mathsf{c}}}\ (\textsc{Nd}) \qquad \dfrac{\varepsilon \vdash_1 P_{\mathsf{b}} : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c}{\varepsilon \vdash_1 \mathsf{b}\langle P_{\mathsf{b}}\rangle : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c + c_{\mathsf{b}}}\ (\textsc{Con}1)}{\varepsilon \vdash_1 P_{\mathsf{b}} : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c + c_{\mathsf{b}}}\ (\textsc{Nd})$$

Starting with the derivation fragment on the left, and then appending the fragment on the right an appropriate number of times, we can derive $\varepsilon \vdash_1 P_{\mathsf{b}} : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c$ for every flag counter c such that $c(\mathsf{a}) = 0$ and $c(\mathsf{c}) = 1$ (where $c(\mathsf{b})$ is arbitrary).

We can derive $\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}\}\!\}] \vdash_1 \lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}$ and $\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}'_{\mathsf{m}} \triangleright c_{\mathsf{a}}$, exactly as in Example 3.9. Let us take $\hat{\sigma}'_R = (\emptyset, \{\!\{0,1\}\!\}, \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}'_{\mathsf{m}}\}\!\} \to o)$. Consider a lambda-term R_3 corresponding to N, namely the unique lambda-term such that $R_3 = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,P_{\mathsf{b}}, R_3\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. By continuing the above derivation concerning P_{b} we obtain:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \mathsf{f} : \hat{\tau}'_{\mathsf{m}} \triangleright \mathbf{0}}\ (\textsc{Var}) \qquad \varepsilon \vdash_1 P_{\mathsf{b}} : (\emptyset, \{\!\{0,1\}\!\}, o) \triangleright c}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \mathsf{f}\,P_{\mathsf{b}} : \hat{\rho}_1^{all} \triangleright c}\ (@)}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \mathsf{nd}\langle \mathsf{f}\,P_{\mathsf{b}}, R_3\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle : \hat{\rho}_1^{all} \triangleright c}\ (\textsc{Nd})}{\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c}\ (\lambda)$$

We also have a derivation fragment that increases the flag counter on the first coordinate:

$$\dfrac{\dfrac{\dfrac{\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c \qquad \varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}\}\!\}] \vdash_1 \lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0} \qquad \varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}'_{\mathsf{m}} \triangleright c_{\mathsf{a}}}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 R_3\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y})) : \hat{\rho}_1^{all} \triangleright c + c_{\mathsf{a}}}\ (@)}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}'_{\mathsf{m}}\}\!\}] \vdash_1 \mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_3\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle : \hat{\rho}_1^{all} \triangleright c + c_{\mathsf{a}}}\ (\textsc{Nd})}{\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c + c_{\mathsf{a}}}\ (\lambda)$$

Notice that, in the last two derivation fragments, the final (λ) rule removes two order-1 markers from the marker multiset of $\hat{\rho}_1^{all}$, so that the marker multiset of $\hat{\sigma}'_R$ contains one order-1 marker. This is because $\hat{\tau}'_{\mathsf{m}}$ provides two order-1 markers. In Example 3.9 $\hat{\tau}_{\mathsf{m}}$ provided three order-1 markers, and, in consequence, $\hat{\sigma}_R$ had no order-1 markers.

By repeating the last derivation fragment, we can derive $\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c$ for every c such that $c(\mathsf{c}) = 1$. We end the derivation as in Example 3.9:

$$\dfrac{\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c \qquad \varepsilon \vdash_1 \lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle : \hat{\tau}_{\mathsf{a}} \triangleright 0 \qquad \varepsilon \vdash_1 \lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle : \hat{\tau}'_{\mathsf{m}} \triangleright c_{\mathsf{a}}}{\varepsilon \vdash_1 \Lambda(\mathcal{G}_3) : \hat{\rho}_2^{all} \triangleright c + c_{\mathsf{a}}}\ (@)$$

□

Remark. In our type derivations and in $\hat{\rho}_m^{all}$ we allow $|\Sigma|$ markers of every positive order, but actually, while solving SUP for a set A, it would be enough to have $|A|$ of them, since we are not interested in counting letters not in A. Similarly, it is not necessary to consider (k, a)-flags for $a \notin A$.

Remark. One can notice that the definition of $Comp_m(M;((F_i,c_i))_{i\in I})$ does not use the value of $M(0)$, so the way how flags are handled does not depend on markers of order 0. In consequence, it is possible to consider a variant of the type system where there are no markers of order 0 at all. A careful analysis can show that such a variant remains correct (i.e., Theorem 3.2 remains true, assuming $m \geq 1$). Nevertheless, the presence of order-0 markers is useful for obtaining the optimal complexity: the condition $M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1$ put on every type triple $(F, M, C_1 \to \ldots \to C_s \to o)$ decreases the number of type triples; see Section 9.1 for more details.

Other type systems. The idea of using intersection types for counting is not completely new. In Parys (2014, 2016) there is a type system that, essentially, allows to estimate the size of the beta-normal form of a (finite) lambda-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 Parys (2012), where we can estimate the number of $\sharp$ 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 (in contrast to our approach, where we have different flags for different orders and letters, 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 (namely, we guess which word $T \in \mathcal{L}(BT(P))$ should be considered), and only after that we want to count something that depends on the chosen values.

Our type system and the type system from Parys (2017b) are, to some extent, motivated by an algorithm of Clemente et al. (2016) solving SUP for schemes. This algorithm works by repeating two kinds of transformations of schemes. The first of them turns the scheme into a scheme generating trees having only a fixed number of branches, one per each letter from A. The branches are chosen nondeterministically out of some tree generated by the original scheme; for every $a \in A$ there is a choice witnessing that a appeared many times in the original tree. Then, such a scheme of the special form is turned into a scheme that is of order lower by one, and generates trees having the same nodes as trees generated by the original scheme, but arranged differently (in particular, the new trees may have again arbitrarily many branches). After finitely many repetitions of this procedure, a scheme of order 0 is obtained, and SUP becomes easily decidable. In some sense we 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 k, we allow to place arbitrarily $|A|$ markers of order k (actually, $|\Sigma|$ of them), which corresponds to the nondeterministic choice of $|A|$ branches in the k-th step of the previous algorithm. We also place some (k,a)-flags, in places that correspond to a-labeled nodes remaining after the k-th step of the previous algorithm.

The idea of having balanced types and unbalanced types (where the former can be used arbitrarily many times, while the latter have to be used exactly once) comes from a type system of Asada and Kobayashi (2016).

Let us compare our type system with the type system introduced in Parys (2017b) in order to solve SUP in the case of $|A| = 1$. The first difference is that we solve the case of multiple letters in A. This is done by replacing a single marker and a single kind of flags of every order by $|\Sigma|$ markers and $|\Sigma|$ kinds of flags, one per each letter of the alphabet Σ. This makes the proofs slightly more complex, but seeing Clemente et al. (2016) and Hague, Kochems, and Ong (2016) it was quite natural that every letter from Σ (or, at least,

every letter from A) requires separate markers and flags.

Conceptually, it was more difficult to establish the optimal complexity. In Parys (2017b) no explicit complexity bound is given, but we can observe that for schemes of order m a direct adaptation of their algorithm to the multiple-letters case works in $(m+3)$-EXPTIME (which drops down to $(m+2)$-EXPTIME for $|A| = 1$ or, more generally, for fixed A); we thus had to save four exponentiations. The previous paper proposes a quite naive algorithm for checking whether there exist type derivations with arbitrarily many flags, which is doubly exponential in the number of type triples, and we replace it by an algorithm that is polynomial in the number of type triples. This saves two exponentiations. Another exponentiation is saved by making the number of order-0 type triples polynomial in $|A|$; this is obtained by making all markers of a fixed order identical (not labeled by a letter, like flags), and by storing information only about one, nondeterministically chosen, flag of every order in flag sets, instead of information about all kinds of flags seen so far. Finally, one more exponentiation is saved by restricting to word-recognizing schemes, and observing that in this case the number of order-1 types can be also made polynomial. This is possible because in every word there is a unique leaf in which an order-0 marker may be placed.

Finally, let us also mention that Asada and Kobayashi (2017) use a slight modification of our type system in order to prove a pumping lemma for languages recognized by higher-order recursion schemes.

4 Finite approximations of infinite lambda-terms

Theorem 3.2 talks about infinite lambda-terms, but the properties described by this theorem concern actually only finite prefixes of these lambda-terms. Moreover, while proving this theorem it is easier to concentrate on finite lambda-terms. For this reason we now formalize the concept of taking a finite approximation of a lambda-term.

We first explain what it means that one lambda-term is an approximation of another lambda-term. This is described by the relation $\preccurlyeq$ defined as the smallest reflexive relation such that:

- $P \preccurlyeq P$ for all lambda-terms P,
- $\lambda x_1^{\alpha_1}.\cdots.\lambda x_s^{\alpha_s}.\mathsf{nd}\langle\rangle \preccurlyeq Q$ whenever Q is of sort $\alpha_1 \to \ldots \to \alpha_s \to o$,
- $a\langle P_1, \ldots, P_r\rangle \preccurlyeq a\langle P'_1, \ldots, P'_r\rangle$ if $P_i \preccurlyeq P'_i$ for all $i \in \{1, \ldots, r\}$,
- $P\,Q \preccurlyeq P'\,Q'$ if $P \preccurlyeq P'$ and $Q \preccurlyeq Q'$, and
- $\lambda x.P \preccurlyeq \lambda x.P'$ if $P \preccurlyeq P'$.

In other words, we allow to replace some subterms Q by lambda-terms of the form $\lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle\rangle$ (where the quantity of variables $x_1, \ldots, x_s$ and their sorts are chosen so that the sort of the lambda-term remains unchanged).

The fact that in Theorem 3.2 it is enough to consider finite approximations of lambda-terms is given by the following two lemmata.

Lemma 4.1. *We can derive a type judgment $\Gamma \vdash_m P : \hat{\tau} \triangleright c$ if and only if for some finite lambda-term P' such that $P' \preccurlyeq P$ we can derive $\Gamma \vdash_m P' : \hat{\tau} \triangleright c$.*

Proof: For the left-to-right implication we recall that type derivations are finite by assumption. We can thus cut off (i.e., replace by $\lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle\rangle$) those subterms of P to which we do not descend while deriving $\Gamma \vdash_m P : \hat{\tau} \triangleright c$. For the opposite implication we observe that it is impossible to derive any type judgment for a lambda-term of the form $\lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle\rangle$, because we cannot apply the (ND) rule to a node constructor without any child. We can thus replace subterms of this form by the actual subterms of P, without altering the type derivation. Details, being easy, are left to the reader. □

Lemma 4.2. *Let P be a closed lambda-term of sort o. For every tree T it is the case that $T \in \mathcal{L}(BT(P))$ if and only if there exists a finite lambda-term P' such that $P' \preccurlyeq P$ and $T \in \mathcal{L}(BT(P'))$.*

The remaining part of this section is devoted to a formal proof of Lemma 4.2. The proof is tedious, but essentially straightforward. The first three lemmata are useful while showing its right-to-left implication.

Lemma 4.3. *If $R' \preccurlyeq R$ and $S' \preccurlyeq S$, then $R'[S'/x] \preccurlyeq R[S/x]$.*

Proof: A trivial coinduction on the structure of R'. □

Lemma 4.4. *If $P' \preccurlyeq P$ and $P' \xrightarrow{h}{}^*_\beta Q'$, then there exists a lambda-term Q such that $Q' \preccurlyeq Q$ and $P \xrightarrow{h}{}^*_\beta Q$.*

Proof: We proceed by induction on the length of the shortest reduction sequence witnessing $P' \xrightarrow{h}{}^*_\beta Q'$; only the base case of a single beta-reduction is interesting, thus assume that $P' \xrightarrow{h}{}_\beta Q'$. In this case $P' = (\lambda x.R')\, S'_0\, S'_1\, \dots\, S'_s$ and $Q' = R'[S'_0/x]\, S'_1\, \dots\, S'_s$ (where $s \geq 0$).

We have two cases. One possibility is that $P = (\lambda x.R)\, S_0\, S_1\, \dots\, S_s$, where $R' \preccurlyeq R$ and $S'_i \preccurlyeq S_i$ for all $i \in \{0, \dots, s\}$. In this case, taking $Q = R[S_0/x]\, S_1\, \dots\, S_s$ we have that $P \xrightarrow{h}{}^*_\beta Q$, and, by Lemma 4.3, $Q' \preccurlyeq Q$.

It is also possible that $R' = \lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle\rangle$. In this case we simply take $Q = P$, and we observe that $Q' = R' \preccurlyeq Q$.[(i)] □

Lemma 4.5. *If P' and P are closed lambda-terms of sort o such that $P' \preccurlyeq P$, and if T is a finite Σ-labeled tree such that $BT(P') \to^n_{\mathsf{nd}} T$, then $BT(P) \to^*_{\mathsf{nd}} T$.*

Proof: The proof is by induction on $|T| + n$. Because $BT(P') \to^n_{\mathsf{nd}} T$, necessarily $BT(P') \neq \mathsf{nd}\langle\rangle$, and thus $P' \xrightarrow{h}{}^*_\beta a\langle P'_1, \dots, P'_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$ and some lambda-terms $P'_1, \dots, P'_r$ such that $BT(P') = a\langle BT(P'_1), \dots, BT(P'_r)\rangle$. Since $P' \preccurlyeq P$, by Lemma 4.4 there exist lambda-terms $P_1, \dots, P_r$ such that $P \xrightarrow{h}{}^*_\beta a\langle P_1, \dots, P_r\rangle$ and $P'_i \preccurlyeq P_i$ for all $i \in \{1, \dots, r\}$. Notice that $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$. We have two cases.

Suppose first that $a \neq \mathsf{nd}$. Then $BT(P') \to^n_{\mathsf{nd}} T$ implies that $T = a\langle T_1, \dots, T_r\rangle$, where for all $i \in \{1, \dots, r\}$ it is the case that $|T_i| < |T|$ and $BT(P'_i) \to^{n_i}_{\mathsf{nd}} T_i$ for some $n_i \leq n$. For every $i \in \{1, \dots, r\}$ the induction assumption implies that $BT(P_i) \to^*_{\mathsf{nd}} T_i$, and thus $BT(P) \to^*_{\mathsf{nd}} T$, as required.

Next, suppose that $a = \mathsf{nd}$. In this case we have $BT(P') \to_{\mathsf{nd}} BT(P'_i) \to^{n-1}_{\mathsf{nd}} T$ for some $i \in \{1, \dots, r\}$. Then $BT(P_i) \to^*_{\mathsf{nd}} T$ by the induction assumption (used for one fixed i only), and we can conclude observing that $BT(P) \to_{\mathsf{nd}} BT(P_i)$. □

The first step needed while proving the left-to-right implication of Lemma 4.2 is to show that every tree from $\mathcal{L}(BT(P))$ can be seen already after performing finitely many beta-reductions from P.

Lemma 4.6. *Let P be a closed lambda-term of sort o, and let T be a finite Σ-labeled tree such that $BT(P) \to^n_{\mathsf{nd}} T$. Then there exists a lambda-term Q such that $P \to^*_\beta Q \to^*_{\mathsf{nd}} T$.*

Proof: The proof is by induction on $|T| + n$. Since $BT(P) \to^n_{\mathsf{nd}} T$, necessarily $BT(P) \neq \mathsf{nd}\langle\rangle$, and thus $P \xrightarrow{h}{}^*_\beta a\langle P_1, \dots, P_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$ and some lambda-terms $P_1, \dots, P_r$ such that $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$. We have two cases.

(i) In the latter case we only know that P is of the form $T\, S_0\, S_1\, \dots\, S_s$, but not necessarily $(\lambda x.R)\, S_0\, S_1\, \dots\, S_s$, so we cannot proceed as in the former case.

Suppose first that $a \neq \mathsf{nd}$. Then $T = a\langle T_1, \ldots, T_r\rangle$, and for all $i \in \{1, \ldots, r\}$ it is the case that $|T_i| < |T|$ and $BT(P_i) \to_{\mathsf{nd}}^{n_i} T_i$ for some $n_i \leq n$. For every $i \in \{1, \ldots, r\}$ the induction assumption gives us a lambda-term Q_i such that $P_i \to_\beta^* Q_i \to_{\mathsf{nd}}^* T_i$. Taking $Q = a\langle Q_1, \ldots, Q_r\rangle$ we obtain $P \to_\beta^* Q \to_{\mathsf{nd}}^* T$, as required.

Next, suppose that $a = \mathsf{nd}$. In this case we have $BT(P) \to_{\mathsf{nd}} BT(P_i) \to_{\mathsf{nd}}^{n-1} T$ for some $i \in \{1, \ldots, r\}$. Then $P_i \to_\beta^* Q_i \to_{\mathsf{nd}}^* T$ for some lambda-term Q_i, by the induction assumption (used for one fixed i only). Taking $Q_j = P_j$ for $j \in \{1, \ldots, r\} \setminus \{i\}$ and $Q = \mathsf{nd}\langle Q_1, \ldots, Q_r\rangle$ we obtain $P \to_\beta^* Q \to_{\mathsf{nd}} Q_i \to_{\mathsf{nd}}^* T$, as required. □

It is convenient to introduce one more relation: we write $P \approx_l P'$ if the lambda-terms P and P' agree up to depth $l \in \mathbb{N}$. Formally, $\approx_l$ is defined by induction on l as the smallest equivalence relation such that:

- if $l = 0$, then $P \approx_l Q$ for all lambda-terms P, Q of the same sort,
- $a\langle P_1, \ldots, P_r\rangle \approx_l a\langle P'_1, \ldots, P'_r\rangle$ if $l > 0$ and $P_i \approx_{l-1} P'_i$ for all $i \in \{1, \ldots, r\}$,
- $P\,Q \approx_l P'\,Q'$ if $l > 0$, and $P \approx_{l-1} P'$, and $Q \approx_{l-1} Q'$, and
- $\lambda x.P \approx_l \lambda x.P'$ if $l > 0$ and $P \approx_{l-1} P'$.

Observe that $P \approx_l P'$ implies $P \approx_k P'$ for $k < l$. Next, we observe that only a finite prefix of the lambda-term Q obtained in Lemma 4.6 is important.

Lemma 4.7. *Let Q be a closed lambda-term of sort o, and let T be a finite Σ-labeled tree such that $Q \to_{\mathsf{nd}}^n T$. Then $BT(Q') \to_{\mathsf{nd}}^* T$ for all lambda-terms Q' such that $Q \approx_{|T|+n} Q'$.*

Proof: Again, the proof is by induction on $|T| + n$. Since $Q \to_{\mathsf{nd}}^n T$ and $|T| + n \geq 1$, necessarily Q and Q' are of the form $a\langle Q_1, \ldots, Q_r\rangle$ and $a\langle Q'_1, \ldots, Q'_r\rangle$, respectively, where $Q_i \approx_{|T|+n-1} Q'_i$ for all $i \in \{1, \ldots, r\}$. Clearly $BT(Q') = a\langle BT(Q'_1), \ldots, BT(Q'_r)\rangle$. We have two cases.

Suppose first that $a \neq \mathsf{nd}$. Then $T = a\langle T_1, \ldots, T_r\rangle$, and for all $i \in \{1, \ldots, r\}$ it is the case that $|T_i| < |T|$ and $Q_i \to_{\mathsf{nd}}^{n_i} T_i$ for some $n_i \leq n$. Since $|T_i| + n_i \leq |T| + n - 1$, we have that $Q_i \approx_{|T_i|+n_i} Q'_i$, hence $BT(Q'_i) \to_{\mathsf{nd}}^* T_i$ by the induction assumption (for all $i \in \{1, \ldots, r\}$). In consequence, $BT(Q') \to_{\mathsf{nd}}^* T$.

Next, suppose that $a = \mathsf{nd}$. Then $Q \to_{\mathsf{nd}} Q_i \to_{\mathsf{nd}}^{n-1} T$ for some $i \in \{1, \ldots, r\}$. Since $Q_i \approx_{|T|+n-1} Q'_i$, by the induction assumption we obtain that $BT(Q'_i) \to_{\mathsf{nd}}^* T$, which together with $BT(Q') \to_{\mathsf{nd}} BT(Q'_i)$ gives us that $BT(Q') \to_{\mathsf{nd}}^* T$, as required. □

The next two lemmata describe what happens during a beta-reduction.

Lemma 4.8. *If $P \approx_l P'$ and $Q \approx_l Q'$ for some $l \in \mathbb{N}$, then also $P[Q/x] \approx_l P'[Q'/x]$.*

Proof: Induction on l. For $l = 0$ the lemma is obvious: $\approx_0$ always holds. When $l > 0$ and $P = R\,S$, then $P' = R'\,S'$ with $R \approx_{l-1} R'$ and $S \approx_{l-1} S'$. By the induction assumption we have $R[Q/x] \approx_{l-1} R'[Q'/x]$ and $S[Q/x] \approx_{l-1} S'[Q'/x]$, and thus $P[Q/x] \approx_l P'[Q'/x]$. The cases when $P = a\langle P_1, \ldots, P_r\rangle$ or $P = \lambda y.Q$ are similar. Finally, when $P = P'$ is a variable, the thesis follows immediately from $Q \approx_l Q'$. □

Lemma 4.9. *If $P \approx_{l+2} P'$ and $P \to_\beta Q$, then for some Q' we have that $P' \to_\beta^* Q'$ and $Q \approx_l Q'$.*

Proof: Induction on l. If $l = 0$, the thesis holds for $Q' = P'$. Next, suppose that $l > 0$ and $P = (\lambda x.R)\,S$ and $Q = R[S/x]$. Then $P' = (\lambda x.R')\,S'$, where $R \approx_l R'$ and $S \approx_{l+1} S'$. Taking $Q' = R'[S'/x]$ we have $P' \to_\beta Q'$, and, by Lemma 4.8, $Q \approx_l Q'$. The remaining case is that $l > 0$ and the redex involved in the beta-reduction $P \to_\beta Q$ is not located on the front of P. Then the thesis follows from the induction

assumption. Let us consider only a representative example: suppose that $P = R\,S$, and $Q = T\,S$, and $R \to_\beta T$. In this case $P' = R'\,S'$ with $R \approx_{l+1} R'$ and $S \approx_{l+1} S'$. The induction assumption gives us T' such that $R' \to_\beta^* T'$ and $T \approx_{l-1} T'$. Thus for $Q' = T'S'$ we have $P' \to_\beta^* Q'$ and $Q \approx_l Q'$. □

We can now conclude the proof of Lemma 4.2.

Proof of Lemma 4.2: The right-to-left implication follows directly from Lemma 4.5. Indeed, recall that we have a finite lambda-term P' such that $P' \preccurlyeq P$ and $T \in \mathcal{L}(BT(P'))$. By the definition of $\mathcal{L}(\cdot)$, the latter means that $BT(P') \to_{\mathsf{nd}}^n T$ for some $n \in \mathbb{N}$, and that T is a finite Σ-labeled tree. In such a situation Lemma 4.5 implies $BT(P) \to_{\mathsf{nd}}^* T$, and thus $T \in \mathcal{L}(BT(P))$, as required.

Let us now prove the opposite implication. We know that $T \in \mathcal{L}(BT(P))$, that is, that T is a finite Σ-labeled tree and $BT(P) \to_{\mathsf{nd}}^n T$ for some $n \in \mathbb{N}$. Then, by Lemma 4.6, there exists a lambda-term Q such that $P \to_\beta^k Q \to_{\mathsf{nd}}^m T$ for some $k, m \in \mathbb{N}$. We now take a finite lambda-term P' such that $P' \preccurlyeq P$ and $P \approx_{2k+m+|T|} P'$; it is easy to obtain such P': we simply need to cut off P at depth $2k + m + |T|$. By applying Lemma 4.9 consecutively to every beta-reduction in the reduction sequence witnessing $P \to_\beta^k Q$ we obtain a lambda-term Q' such that $P' \to_\beta^* Q'$ and $Q \approx_{m+|T|} Q'$. Next, Lemma 4.7 implies that $BT(Q') \to_{\mathsf{nd}}^* T$. Since $BT(P')$ equals $BT(Q')$ (this is the beta-normal form of these finite lambda-terms), we obtain $T \in \mathcal{L}(BT(P'))$, as required. □

5 Properties of type judgments

Before actually proving Theorem 3.2 in the next two sections, we state here some properties of those type judgments that can be derived in our type system.

We start by a simple lemma, that follows directly from rules of the type system. This lemma is used implicitly later.

Lemma 5.1. *If we can derive $\Gamma \vdash_m R : \hat\tau \triangleright c$, and x is not free in R, then $\Gamma(x) = \mathbf{0}$.* □

Next, in Lemma 5.2, we formalize the intuition that the marker multiset of a type judgment includes all markers provided by free variables (which are described in the type environment).

Lemma 5.2. *Suppose that we can derive $\Gamma \vdash_m R : \hat\tau \triangleright c$. Then $\mathsf{Mk}(\Gamma) \le \mathsf{Mk}(\hat\tau)$.*

Proof: Fix some derivation of $\Gamma \vdash_m R : \hat\tau \triangleright c$; the proof is by induction on the structure of this derivation. We analyze the shape of R.

Suppose first that $R = x$. The (VAR) rule says that $\Gamma = \varepsilon[x \mapsto \{\!|\hat\tau'|\!\}]$ for $\hat\tau'$ such that $\mathsf{Mk}(\hat\tau') \le \mathsf{Mk}(\hat\tau)$, which implies that $\mathsf{Mk}(\Gamma) \le \mathsf{Mk}(\hat\tau)$.

In the case when $R = a\langle\rangle$ for $a \neq \mathsf{nd}$, the (CON0) rule implies that $\Gamma = \varepsilon$, hence $\mathsf{Mk}(\Gamma) \le \mathsf{Mk}(\hat\tau)$.

Next, suppose that $R = \lambda x.P$. Let $\Gamma[x \mapsto C'] \vdash_m P : \hat\tau' \triangleright c$ be the premiss of the final (λ) rule, and let $C \to \tau$ be the type appearing in the type triple $\hat\tau$. By conditions of the rule we have $C' \sqsubseteq C$ and $\mathsf{Mk}(\hat\tau) = \mathsf{Mk}(\hat\tau') - \mathsf{Mk}(C)$. While writing $\Gamma[x \mapsto C']$ we mean that $\Gamma(x) = \mathbf{0}$, so $\mathsf{Mk}(\Gamma) = \mathsf{Mk}(\Gamma[x \mapsto C']) - \mathsf{Mk}(C')$. The condition $C' \sqsubseteq C$ implies that $\mathsf{Mk}(C) = \mathsf{Mk}(C')$, and the induction assumption ensures that $\mathsf{Mk}(\Gamma[x \mapsto C']) \le \mathsf{Mk}(\hat\tau')$. Putting this together we obtain $\mathsf{Mk}(\Gamma) \le \mathsf{Mk}(\hat\tau)$.

Finally, suppose that $R = a\langle P\rangle$ for $a \neq \mathsf{nd}$, or $P = \mathsf{nd}\langle P_1, \dots, P_r\rangle$, or $R = P\,Q$. Let $\hat\tau_1, \dots, \hat\tau_s$ be the type triples derived in premisses of the final rule (which is either (CON1), or (ND), or (@)), and let $\Gamma_1, \dots, \Gamma_s$ be the type environments used there. Each of the three possible rules ensures that $\mathsf{Mk}(\hat\tau) = \mathsf{Mk}(\hat\tau_1) + \dots +$

$\mathsf{Mk}(\hat{\tau}_s)$ and $\Gamma = \Gamma_1 \sqcup \cdots \sqcup \Gamma_s$. The induction assumption gives us inequalities $\mathsf{Mk}(\Gamma_i) \leq \mathsf{Mk}(\hat{\tau}_i)$ for all $i \in \{1, \ldots, s\}$. It follows that $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$. □

The next important property of our type system is given in Lemma 5.3.

Lemma 5.3. *If a type judgment $\Delta \vdash_m R : \hat{\sigma} \triangleright d$ is used in a derivation of $\Gamma \vdash_m S : \hat{\tau} \triangleright c$, where $\mathsf{Mk}(\hat{\tau})(m) = 0$ and $ord(S) \leq m$, then $\mathsf{Mk}(\hat{\sigma})(m) = 0$.*

Proof: We say that a type triple $\hat{\sigma} = (F, M, C_1 \to \ldots \to C_s \to o)$ is *m-clear* if $(M + \sum_{i=1}^{s} \mathsf{Mk}(C_i))(m) = 0$. It is enough to prove that, in the considered derivation, there are only type judgments with m-clear type triples; then the statement of the lemma follows immediately.

We first notice that if $\hat{\sigma}$ is derived for a lambda-term having sort α of order at most m, and $\mathsf{Mk}(\hat{\sigma})(m) = 0$, then $\hat{\sigma}$ is m-clear. Indeed, let us write $\hat{\sigma} = (F, M, C_1 \to \ldots \to C_s \to o)$. For $i \in \{1, \ldots, s\}$ by definition we have $C_i \in \mathcal{TC}^{\alpha_i}$, where $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o$; type triples in C_i belong to $\mathcal{TT}^{\alpha_i}_{ord(\alpha_i)}$, so $\mathsf{Mk}(C_i)$ is $ord(\alpha_i)$-bounded, and because $ord(\alpha_i) < ord(\alpha) \leq m$, we obtain $\mathsf{Mk}(C_i)(m) = 0$ as needed. In particular it follows that the type triple $\hat{\tau}$ derived at the end is m-clear.

It remains to prove that if a conclusion of some rule derives an m-clear type triple, then all its premisses as well. Let $\Delta \vdash_m R : (F, M, \sigma) \triangleright d$ be the considered conclusion, where $\sigma = C_1 \to \ldots \to C_s \to o$. We have several cases depending on the shape of R.

If $R = x$ or $R = a\langle\rangle$, the thesis is immediate, as there are no premisses. If $R = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$, then the (ND) rule is used, so the type triple derived in the premiss is the same as in the conclusion. If $R = a\langle P\rangle$ for $a \neq \mathsf{nd}$ then, on the one hand, $s = 0$ and, on the other hand, the (CON1) rule is used, and it has a premiss $\Delta \vdash_m P : (F', M, o) \triangleright d'$ with an unchanged marker multiset.

Suppose that $R = \lambda x.P$. Then the (λ) rule is used, and it has a premiss $\Delta' \vdash_m P : (F, M', \sigma') \triangleright d$, where $\sigma' = C_2 \to \ldots \to C_s \to o$ and $M' = M + \mathsf{Mk}(C_1)$. We thus have $(M' + \sum_{i=2}^{s} \mathsf{Mk}(C_i))(m) = (M + \sum_{i=1}^{s} \mathsf{Mk}(C_i))(m) = 0$.

Finally, suppose that $R = P\,Q$. Let $\Delta' \vdash_m P : (F', M', C_0 \to \sigma) \triangleright d'$ and $\Delta_i \vdash_m Q : (F_i, M_i, \sigma_i) \triangleright d_i$ for $i \in I$ be the premisses of the considered rule, which is (@). The rule implies that $M = M' + \sum_{i \in I} M_i$, so $M'(m) = 0$ and $M_i(m) = 0$ for all $i \in I$. It also implies that $ord(Q) \leq m$, so the type triples (F_i, M_i, σ_i) derived for Q are m-clear (as observed at the beginning). Moreover, the marker multisets in type triples in C_0 are $M_i{\restriction}_{\leq ord(Q)}$, so $\mathsf{Mk}(C_0)(m) = 0$, and thus also $(F', M', C_0 \to \sigma)$ is m-clear. □

Out of Lemma 5.3 we easily deduce the following lemma.

Lemma 5.4. *Suppose that we can derive $\Gamma \vdash_m S : \hat{\tau} \triangleright c$, where $\mathsf{Mk}(\hat{\tau})(m) = 0$ and $ord(S) \leq m$. Then $c = \mathbf{0}$.*

Proof: Suppose to the contrary that $c \neq \mathbf{0}$. Then for some rule used in the derivation, its conclusion $\Delta \vdash_m R : (F, M, \sigma) \triangleright d$ has a nonzero flag counter d, but flag counters in all premisses are $\mathbf{0}$. This is possible only in the following rules: (CON0), (CON1), or (@). In these rules we have $(F, d) \in Comp_m(M; \cdots)$, where by Lemma 5.3 we have $M(m) = 0$. To the $Comp_m$ predicate we pass pairs (F_i, c_i) corresponding to particular premisses; for them we know that $c_i = \mathbf{0}$ and that F_i is m-bounded. In the (CON0) and (CON1) rules we additionally pass a pair $(F_i, c_i) = (\{(1, a)\}, \mathbf{0})$. In these rules $\sigma = o$, so $M(0) = 1$ by the definition of a type triple (more generally, if $\sigma = C_1 \to \ldots \to C_s \to o$, then $M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_1)(0) = 1$), which implies that $m \geq 1$ (recall that $M(m) = 0 \neq 1$). In consequence, for this pair it is also the case that $c_i = \mathbf{0}$ and that F_i is m-bounded. It follows that the numbers $f'_{m+1,a}$ and $f_{m+1,a}$ appearing in the definition of $Comp_m$ are 0, and thus necessarily $d = \mathbf{0}$, contrary to our assumption. □

One may suspect that Lemma 5.4 can be generalized to lower orders, that is, that whenever we can derive $\Gamma \vdash_m S : (F, M, \tau) \triangleright c$ with $M(k) = 0$ for all $k \geq ord(S)$, then $F\restriction_{>ord(S)} = \emptyset$. The justification of such a statement would be as those of Lemma 5.4: flags of order $k+1 > ord(S)$ are created only when a marker of order $k \geq ord(S)$ is visible, while such markers are not provided neither in the derivation itself (since $M(k) = 0$) nor in the the arguments of the lambda-term. Life is not so simple, however: it may be the case that Γ simply provides some flag of order greater than $ord(S)$. This is illustrated by the following example.

Example 5.1. In this example we assume that $\Sigma = \{\mathsf{a}\}$; then $\hat{\rho}_1^{all} = (\emptyset, \{\!\{0,1\}\!\}, o)$. Denote

$$\hat{\tau}_{\mathsf{y}} = (\{(1,\mathsf{a})\}, \mathbf{0}, \mathbf{0} \to o) \qquad \text{and} \qquad \hat{\sigma} = (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!\{\hat{\tau}_{\mathsf{y}}\}\!\} \to \mathbf{0} \to o)\,,$$

and consider the following type derivation, in which x is of sort o, y is of sort $o \to o$, and z is of sort $(o \to o) \to o \to o$. As in the previous examples, c_{a} is the flag counter such that $c_{\mathsf{a}}(\mathsf{a}) = 1$.

$$\dfrac{\dfrac{\dfrac{}{\varepsilon \vdash_1 \mathsf{a}\langle\rangle : \hat{\rho}_1^{all} \triangleright c_{\mathsf{c}}}\,(\textsc{Con}0)}{\varepsilon \vdash_1 \lambda\mathsf{z}.\mathsf{a}\langle\rangle : (\emptyset, \{\!\{0,1\}\!\}, \{\!\{\hat{\sigma}\}\!\} \to o) \triangleright c_{\mathsf{c}}}\,(\lambda) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{y} \mapsto \{\!\{\hat{\tau}_{\mathsf{y}}\}\!\}] \vdash_1 \mathsf{y} : \hat{\tau}_{\mathsf{y}} \triangleright \mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{y} \mapsto \{\!\{\hat{\tau}_{\mathsf{y}}\}\!\}] \vdash_1 \mathsf{y}\,\mathsf{x} : (\{(1,\mathsf{a})\}, \mathbf{0}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{y} \mapsto \{\!\{\hat{\tau}_{\mathsf{y}}\}\!\}] \vdash_1 \lambda\mathsf{x}.\mathsf{y}\,\mathsf{x} : \hat{\tau}_{\mathsf{y}} \triangleright \mathbf{0}}\,(\lambda)}{\varepsilon \vdash_1 \lambda\mathsf{y}.\lambda\mathsf{x}.\mathsf{y}\,\mathsf{x} : \hat{\sigma} \triangleright \mathbf{0}}\,(\lambda)}{\varepsilon \vdash_1 (\lambda\mathsf{z}.\mathsf{a}\langle\rangle)\,(\lambda\mathsf{y}.\lambda\mathsf{x}.\mathsf{y}\,\mathsf{x}) : \hat{\rho}_1^{all} \triangleright c_{\mathsf{c}}}\,(@)$$

The type judgment concerning the subterm $\mathsf{y}\,\mathsf{x}$ is of the considered "illegal" form: it provides a flag of order 1, but does not use any markers. This means two things: first, that such a type judgment can be derived, and second, that it can be used in a derivation concerning a closed lambda-term of sort o. We notice, however, that this type judgment could appear in the whole derivation only because it is actually ignored ($\lambda\mathsf{z}.\mathsf{a}\langle\rangle$ ignores its argument z); otherwise, it would be necessary to derive $\hat{\tau}_{\mathsf{y}}$ for some subterm that would be given as y to $\lambda\mathsf{y}.\lambda\mathsf{x}.\mathsf{y}\,\mathsf{x}$, and this would be impossible, since we cannot create a flag of order 1 without using markers. □

We thus have to generalize Lemma 5.4 in a more subtle way, having in mind the above issues. To this end, we proceed in a minimalistic way: in Lemma 5.5 we prove what is really useful for us, although it might be effortless to prove a slightly stronger result. While formulating this lemma we need the following definition. Consider a use of the (@) rule that derives a type judgment $\Gamma \vdash_m P\,Q : \hat{\tau} \triangleright c$. We say that this use of the (@) rule is *wild* if it has a premiss $\Gamma' \vdash_m Q : (F, \mathbf{0}, \sigma) \triangleright c'$ such that for some $(k, a) \in F\restriction_{>ord(Q)}$ it is the case that $\mathsf{Mk}(\hat{\tau})(l) > 0$ for all $l \in \{k, k+1, \ldots, m\}$. A type derivation is *wild* if, at some moment, it uses the (@) rule in a wild way.

Lemma 5.5. *There is no wild derivation of* $\Gamma \vdash_m P : \hat{\rho}_m^{all} \triangleright c$*, where* P *is a closed lambda-term of sort* o *and of complexity at most* $m+1$*.*

This lemma is proven in Section 7. We remark that we do not use this lemma in Sections 6 and 7, only in Section 9. Right now we only prove the following auxiliary lemma.

Lemma 5.6. *There is no wild derivation of* $\Gamma \vdash_m S : \hat{\tau} \triangleright c$ *if* $\mathsf{Mk}(\hat{\tau})(m) = 0$ *and* $ord(S) \leq m$*.*

Proof: Suppose that some use of the (@) rule is wild in a derivation of this type judgment. Let $\Delta \vdash_m P\,Q : \hat{\sigma} \triangleright d$ be the conclusion of this rule. The wildness condition requires in particular that $\mathsf{Mk}(\hat{\sigma})(m) > 0$, but by Lemma 5.3 we have $\mathsf{Mk}(\hat{\sigma})(m) = 0$, so all this could not happen. □

6 Completeness

In this section we prove the left-to-right implication of Theorem 3.2. We divide the proof into the following four lemmata. Recall that $P \to_{\beta(k)} Q$ denotes a beta-reduction of order k, that is, a reduction on a redex $(\lambda x.R)\, S$ with $ord(x) = k$.

Lemma 6.1. *Let P be a finite closed lambda-term of sort o and of complexity at most n. Then there exist lambda-terms $Q_n, Q_{n-1}, \ldots, Q_0$ such that $P = Q_n$, and for every $k \in \{0, \ldots, n-1\}$, Q_k is of complexity at most k and $Q_{k+1} \to^*_{\beta(k)} Q_k$, and $Q_0 = BT(P)$.*

Lemma 6.2. *Suppose that $T \in \mathcal{L}(P)$ is a word, and that $c\colon \Sigma \to \mathbb{N}$ is such that for every $a \in \Sigma$, $c(a)$ is the number of occurrences of a in T. Then we can derive $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$.*

Lemma 6.3. *Suppose that $P \to_{\beta(m)} Q$, where $m \geq 0$. If we can derive $\Gamma \vdash_m Q : \hat{\tau} \triangleright c$, then we can also derive $\Gamma \vdash_m P : \hat{\tau} \triangleright c$.*

Lemma 6.4. *If we can derive $\varepsilon \vdash_{m-1} P : \hat{\rho}_{m-1}^{all} \triangleright c$, then we can also derive $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c'$ for some c' such that $c'(a) \geq \left\lfloor \frac{1}{|\Sigma|} \log_2 c(a) \right\rfloor$ for all $a \in \Sigma$.*

Now the left-to-right implication of Theorem 3.2 easily follows. Indeed, take a word-recognizing closed lambda-term P of sort o and of complexity at most $m+1$ and a set $A \subseteq \Sigma$ such that $SUP_A(\mathcal{L}(BT(P)))$ holds, and take any $n \in \mathbb{N}$. Let us denote $f_0(l) = l$ and $f_k(l) = \left\lfloor \frac{1}{|\Sigma|} \log_2 f_{k-1}(l) \right\rfloor$ for $k \geq 1$. We can find a number n' such that $f_m(n') \geq n$, as well as a word $T \in \mathcal{L}(BT(P))$ such that every letter from A appears in T at least n' times. We first apply Lemma 4.2, obtaining a finite lambda-term P' such that $P' \preccurlyeq P$ and $T \in \mathcal{L}(BT(P'))$. Clearly the complexity of P' remains at most $m+1$. Then we apply Lemma 6.1 to P', obtaining lambda-terms $Q_{m+1}, Q_m, \ldots, Q_0$ with $T \in \mathcal{L}(Q_0) = \mathcal{L}(BT(P'))$. By Lemma 6.2 we can derive $\varepsilon \vdash_0 Q_0 : \hat{\rho}_0^{all} \triangleright c_0$ with $c_0(a) \geq n' = f_0(n')$ for all $a \in A$. Next, we repeatedly apply Lemma 6.3 to every beta-reduction (of order 0) between Q_1 and Q_0, obtaining a derivation of $\varepsilon \vdash_0 Q_1 : \hat{\rho}_0^{all} \triangleright c_0$. Then, consecutively for every $k \in \{1, \ldots, m\}$ we perform two steps. First, we apply Lemma 6.4, obtaining a derivation of $\varepsilon \vdash_k Q_k : \hat{\rho}_k^{all} \triangleright c_k$ for some c_k such that $c_k(a) \geq f_1(f_{k-1}(n')) = f_k(n')$ for all $a \in A$. Then, we repeatedly apply Lemma 6.3 to every beta-reduction (of order k) between Q_{k+1} and Q_k, obtaining a derivation of $\varepsilon \vdash_k Q_{k+1} : \hat{\rho}_k^{all} \triangleright c_k$. We end up with a derivation of $\varepsilon \vdash_m P' : \hat{\rho}_m^{all} \triangleright c_m$, where $c_m(a) \geq f_m(n') \geq n$ for all $a \in A$. Using Lemma 4.1 we can convert it into a derivation of $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_m$, as needed.

Lemma 6.1 comes from our previous work (Parys, 2018a, Lemma 11). In the remaining part of this section we prove Lemmata 6.2-6.4.

Remark. We notice that Lemma 6.1 would be false if we have allowed lambda-terms involving non-homogeneous sorts. For example, in a lambda-term of the form $(\lambda x.\lambda y.P)\, Q\, R$ with $ord(x) = 0$ and $ord(y) = 1$ we have to perform a beta-reduction of order 0 concerning x before a beta-reduction of order 1 concerning y. Homogeneity of sorts should not be seen, however, as a miracle that is necessary to construct the whole type system considered in the paper; it is rather an assumption made for technical convenience. Indeed, Lemma 6.1 would work also for lambda-terms involving non-homogeneous sorts if we have defined the order of a beta-reduction $(\lambda x.R)\, S \to_\beta R[S/x]$ as $ord(\lambda x.R) - 1$, not as $ord(x)$ (notice that these two numbers coincide for homogeneous sorts). However then it would be necessary to alter the definition of a type environment (and similarly the (VAR) rule): $\Gamma(x)$ should not be $ord(x)$-bounded, but rather $(ord(\lambda x.R) - 1)$-bounded, where $\lambda x.R$ is the superterm binding the variable x. This would be

uncomfortable, as $ord(\lambda x.R) - 1$ is contextual information, not determined by x itself. For this reason we prefer to restrict ourselves to homogeneous sorts.

6.1 Proof of Lemma 6.2

Basically, in order to prove Lemma 6.2 we simply apply the rules of our type system, namely the rules (CON0), (CON1), and (ND), in every nd-labeled node choosing the subtree in which T continues; then the flag counter computes exactly the number of occurrences of every letter in T.

Formally, we proceed by induction on $|T| + n$, where n is the smallest number such that $P \to_{\mathsf{nd}}^n T$ (recall that $T \in \mathcal{L}(P)$ by definition means that $P \to_{\mathsf{nd}}^n T$ for some $n \in \mathbb{N}$, that T is finite, and that it is Σ-labeled). Recall that we intend to derive $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$, where c stores the number of occurrences of particular letters in T.

We have three possibilities. One possibility is that $P = \mathsf{nd}\langle P_1, \dots, P_r\rangle$. In this case, the reduction sequence witnessing $P \to_{\mathsf{nd}}^n T$ starts with $P \to_{\mathsf{nd}} P_i$ for some $i \in \{1, \dots, r\}$, and then we have a reduction sequence witnessing $P_i \to_{\mathsf{nd}}^{n-1} T$. The induction assumption gives us a derivation of $\varepsilon \vdash_0 P_i : \hat{\rho}_0^{all} \triangleright c$, where, for all $a \in \Sigma$, $c(a)$ is the number of occurrences of a in T. To this type judgment we apply the (ND) rule, deriving $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$, as needed.

Another possibility is that $P = b\langle\rangle$ for some $b \in \Sigma$. Then necessarily $T = b\langle\rangle$, so $c(b) = 1$ and $c(a) = 0$ for all $a \in \Sigma \setminus \{b\}$. We have that $(\emptyset, c) \in Comp_0(\{\!|0|\!\}; (\{(1,b)\}, \mathbf{0}))$, so the (CON0) rule derives $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$. We recall here that $\mathsf{Mk}(\hat{\rho}_0^{all}) = \{\!|0|\!\}$, and that the flag set in $\hat{\rho}_0^{all}$ is $\emptyset$.

Because $P \to_{\mathsf{nd}}^* T$, and T is a word, the only remaining case is that $P = b\langle P_1\rangle$ for some $b \in \Sigma$. Then necessarily $T = b\langle T_1\rangle$, and out of the reduction sequence witnessing $P \to_{\mathsf{nd}}^n T$ we can extract a reduction sequence witnessing $P_1 \to_{\mathsf{nd}}^n T_1$. By the induction assumption we know that we can derive $\varepsilon \vdash_0 P_1 : \hat{\rho}_0^{all} \triangleright c_1$, where $c_1 \colon \Sigma \to \mathbb{N}$ is such that, for all $a \in \Sigma$, $c_1(a)$ is the number of occurrences of a in T_1. Obviously, $c(a) = c_1(a)$ for $a \in \Sigma \setminus \{b\}$, and $c(b) = 1 + c_1(b)$. Directly from the definition of the $Comp_0$ predicate it follows that $(\emptyset, c) \in Comp_0(\{\!|0|\!\}; (\{(1,b)\}, \mathbf{0}), (\emptyset, c_1))$. By applying the (CON1) rule we obtain $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$, as needed.

6.2 Proof of Lemma 6.3

Before going into details, let us sketch the proof. Knowing that we can derive $\Gamma \vdash_m Q : \hat{\tau} \triangleright c$, we want to derive the same for a given lambda-term P such that $P \to_{\beta(m)} Q$. Let us 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 $\Gamma \vdash_m Q : \hat{\tau} \triangleright c$ we identify the set I of places (nodes) where we derive a type for S substituted for x. For $i \in I$, let $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ be the type judgment in i. We change the nodes in I into leaves, where we instead derive $\varepsilon[x \mapsto \{\!|\hat{\sigma}_i|\!\}] \vdash_m x : \hat{\sigma}_i \triangleright \mathbf{0}$ (here we need to know that $ord(x) = m$, since a type environment should map x into an $ord(x)$-bounded triple container, while $\hat{\sigma}_i$ is m-bounded). It is tedious but straightforward to repair the rest of the derivation, by changing type environments, replacing S by x in lambda-terms, and decreasing flag counters. In this way we obtain derivations of $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and a derivation of $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$, where[(ii)] $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$, and $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$, and $c = e + \sum_{i\in I} d_i$. To the latter type judgment we apply the (λ) rule, and then we merge it with the type judgments for S using the (@) rule, which results in a derivation of $\Gamma \vdash_m P : \hat{\tau} \triangleright c$ (where again we use the fact that $ord(S) = ord(x) = m$). We remark that different $i \in I$ may give identical type judgments for S; this is absolutely allowed in the (@) rule. We

[(ii)] Recall that whenever we write $\Upsilon[x \mapsto \dots]$, we implicitly assume that $\Upsilon(x) = \mathbf{0}$.

also need to know that $\{\!|\hat{\sigma}_i \mid i \in I|\!\}$ is indeed a triple container, that is, that every unbalanced type triple appears as $\hat{\sigma}_i$ for at most $|\Sigma|$ indices $i \in I$; this is a consequence of Lemma 5.2.

We now come to a lemma that splits a type derivation concerning $R[S/x]$ into parts concerning R and concerning S.

Lemma 6.5. *Suppose that we can derive $\Gamma \vdash_m R[S/x] : \hat{\tau} \triangleright c$, where $ord(x) = m$. Then, for some finite set I, we can derive $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$, where $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$, and $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$, and $c = e + \sum_{i\in I} d_i$.*

Proof: The proof is by induction on the structure of some fixed derivation of $\Gamma \vdash_m R[S/x] : \hat{\tau} \triangleright c$.

One possibility is that x is not free in R. Then $R[S/x] = R$, and $\Gamma(x) = \mathbf{0}$. We can take $I = \emptyset$, and $\Upsilon^x = \Upsilon = \Gamma$, and $e = c$. We need to derive the type judgment $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$, but it actually equals the type judgment that we can derive by assumption.

In the sequel we assume that x is free in R. In particular, R is not of the form $a\langle\rangle$, nor R is a variable other than x. We have several cases depending on the shape of R.

Suppose first that $R = x$ is a variable. Then we take $I = \{1\}$, and $(\Delta_1, \hat{\sigma}_1, d_1) = (\Gamma, \hat{\tau}, c)$, and $\Upsilon = \varepsilon$, and $e = \mathbf{0}$. Obviously $\Gamma = \Upsilon \sqcup \Delta_1$ and $c = e + d_1$. Because $\hat{\sigma}_1$ is m-bounded, and $ord(x) = m$, we have that $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_1|\!\}]$ is a valid type environment. We can derive $\Delta_1 \vdash_m S : \hat{\sigma}_1 \triangleright d_1$ by assumption, and $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$ using the (VAR) rule.

Next, suppose that $R = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$. The original derivation ends with the (ND) rule whose premiss is $\Gamma \vdash_m P_k[S/x] : \hat{\tau} \triangleright c$ for some $k \in \{1, \ldots, r\}$. By applying the induction assumption for this premiss, we obtain derivations almost as required; we only need to apply again the (ND) rule to the obtained derivation $\Upsilon^x \vdash_m P_k : \hat{\tau} \triangleright e$

Next, suppose that $R = \lambda y.P$. We have $y \neq x$ (because x is free in R), and, as always during a substitution, we assume (by performing alpha-conversion) that y is not free in S. The original derivation ends with the (λ) rule, whose premiss is $\Gamma[y \mapsto C'] \vdash_m P[S/x] : \hat{\tau}' \triangleright c$. We apply the induction assumption to this premiss, and we obtain a derivation of $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and of $\Upsilon^x[y \mapsto C''] \vdash_m P : \hat{\tau}' \triangleright e$, where $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$, and $\Gamma[y \mapsto C'] = \Upsilon[y \mapsto C''] \sqcup \bigsqcup_{i\in I} \Delta_i$, and $c = e + \sum_{i\in I} d_i$. Notice that $\Delta_i(y) = \mathbf{0}$ for all $i \in I$, because y is not free in S; it follows that $C'' = C'$ and $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$. To the type judgment $\Upsilon^x[y \mapsto C'] \vdash_m P : \hat{\tau}' \triangleright e$ we apply again the (λ) rule, which gives $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$.

Another possibility is that $R = a\langle P\rangle$, where $a \neq \mathsf{nd}$. Then the original derivation ends with the (CON1) rule, whose premiss is $\Gamma \vdash_m P[S/x] : \hat{\tau}_1 \triangleright c_1$. We apply the induction assumption to this premiss. We obtain a derivation of $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and of $\Upsilon^x \vdash_m P : \hat{\tau}_1 \triangleright e_1$, where $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$, and $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$, and $c_1 = e_1 + \sum_{i\in I} d_i$. A side condition of the (CON1) rule says that $(F, c) \in Comp_m(M; (\{(1, a)\}, \mathbf{0}), (F_1, c_1))$ for appropriate arguments M, F, F_1. Taking $e = c + (e_1 - c_1)$ we also have that $(F, e) \in Comp_m(M; (\{(1, a)\}, \mathbf{0}), (F_1, e_1))$. Thus, we can apply the (CON1) rule again, deriving $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$. Simultaneously we observe that $c = e + \sum_{i\in I} d_i$.

Finally, suppose that $R = P\,Q$. The original derivation ends with the (@) rule, whose premisses are $\Gamma_0 \vdash_m P[S/x] : \hat{\tau}_0 \triangleright c_0$ and $\Gamma_j \vdash_m Q[S/x] : \hat{\tau}_j \triangleright c_j$ for $j \in J$, where we assume that $0 \notin J$. We apply the induction assumption to all these premisses. Assuming without loss of generality that the resulting sets I_j are disjoint, and taking $I = \bigcup_{j\in\{0\}\cup J} I_j$, we obtain a derivation of $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and of $\Upsilon^x_0 \vdash_m P : \hat{\tau}_0 \triangleright e_0$, and of $\Upsilon^x_j \vdash_m Q : \hat{\tau}_j \triangleright e_j$ for every $j \in J$, where, for every $j \in \{0\} \cup J$, we have $\Upsilon^x_j = \Upsilon_j[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I_j|\!\}]$, and $\Gamma_j = \Upsilon_j \sqcup \bigsqcup_{i\in I_j} \Delta_i$, and $c_j = e_j + \sum_{i\in I_j} d_i$. Let $\Upsilon = \bigsqcup_{j\in\{0\}\cup J} \Upsilon_j$. Because $\Gamma = \bigsqcup_{j\in\{0\}\cup J} \Gamma_j$, we see that $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$. For $j \in \{0\} \cup J$

we have by Lemma 5.2 that $\mathsf{Mk}(\Upsilon^x_j) \le \mathsf{Mk}(\hat\tau_j)$, hence in particular $\sum_{i\in I_j}\mathsf{Mk}(\hat\sigma_i) \le \mathsf{Mk}(\hat\tau_j)$. Since $\mathsf{Mk}(\hat\tau) = \sum_{j\in\{0\}\cup J}\mathsf{Mk}(\hat\tau_j)$ (which follows from the (@) rule) we obtain that $\sum_{i\in I}\mathsf{Mk}(\hat\sigma_i) \le \mathsf{Mk}(\hat\tau)$. Because $\mathsf{Mk}(\hat\tau)$ is a marker multiset (i.e., contains every marker at most $|\Sigma|$ times), we can deduce that every unbalanced type triple appears as $\hat\sigma_i$ for at most $|\Sigma|$ indices $i \in I$, and thus $\{\!\{\hat\sigma_i \mid i \in I\}\!\}$ is a valid triple container and $\Upsilon^x = \Upsilon[x \mapsto \{\!\{\hat\sigma_i \mid i \in I\}\!\}]$ is a valid type environment. By applying the (@) rule again, we derive $\Upsilon^x \vdash_m R : \hat\tau \triangleright e$, where $e = c + \sum_{j\in\{0\}\cup J}(e_j - c_j)$. Side conditions of this rule remain satisfied, since the derived type triples are the same as in the original derivation (only the flag counters in particular premisses are modified by $e_j - c_j$, which modifies the flag counter in the conclusion by $\sum_{j\in\{0\}\cup J}(e_j - c_j)$). Moreover, $c = e + \sum_{i\in I} d_i$. □

Before actually proving Lemma 6.3, we need an auxiliary lemma concerning the $Comp_m$ predicate.

Lemma 6.6. *If $F \in \mathcal{F}_m$, and $M \in \mathcal{M}_m$, and $M(k) = 0$ for all $(k,a) \in F$ then*

$$(F', c) \in Comp_m(M; (F, e), ((\emptyset, d_i))_{i\in I}) \qquad \Leftrightarrow \qquad F' \subseteq F \wedge c = e + \sum_{i\in I} d_i\,.$$

Proof: Consider the numbers $f_{k,a}$ and $f'_{k,a}$ appearing in the definition of the $Comp_m$ predicate. Looking at them consecutively for $k = 1, \ldots, m+1$ we notice that $f'_{k,a} = 0$ and $f_{k,a} = |F \cap \{(k,a)\}|$. Indeed, $f'_{k,a} = 0$ implies $f_{k,a} = |F \cap \{(k,a)\}|$, and if $k = 1$ or $M(k-1) = 0$, we have $f'_{k,a} = 0$, while if $k > 1$ and $M(k-1) > 0$, we have $f'_{k,a} = f_{k-1,a} = |F \cap \{(k-1,a)\}| = 0$, because $(k-1,a) \in F$ implies $M(k-1) = 0$ by assumption. It follows that $F = \{(k,a) \mid f_{k,a} > 0\}$, and that $f_{m+1,a} = |F \cap \{(m+1,a)\}| = 0$ for all $a \in \Sigma$ (since F is m-bounded). Finally, recall that $(F', c) \in Comp_m(M; (F, e), ((\emptyset, d_i))_{i\in I})$ if and only if $F' \subseteq \{(k,a) \mid f_{k,a} > 0\}$ (i.e., $F' \subseteq F$) and $c(a) = f_{m+1,a} + e(a) + \sum_{i\in I} d_i(a)$ for all $a \in \Sigma$ (i.e., $c = e + \sum_{i\in I} d_i$). □

Proof of Lemma 6.3: Recall that we are given a derivation of $\Gamma \vdash_m Q : \hat\tau \triangleright c$, and a beta-reduction $P \to_\beta Q$ that is of order m, and our goal is to derive $\Gamma \vdash_m P : \hat\tau \triangleright c$.

Suppose first that $P = (\lambda x.R)\,S$ and $Q = R[S/x]$, where $ord(x) = m$. From Lemma 6.5 we obtain a derivation of $\Delta_i \vdash_m S : \hat\sigma_i \triangleright d_i$ for every $i \in I$ (for some set I), and a derivation of $\Upsilon[x \mapsto C] \vdash_m R : \hat\tau \triangleright e$, where $C = \{\!\{\hat\sigma_i \mid i \in I\}\!\}$, and $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I}\Delta_i$, and $c = e + \sum_{i\in I} d_i$. Let us write $\hat\tau = (F, M, \tau)$, and $\hat\sigma_i = (F_i, M_i, \sigma_i)$ for $i \in I$. To the type judgment $\Upsilon[x \mapsto C] \vdash_m R : \hat\tau \triangleright e$ we apply the (λ) rule, deriving $\Upsilon \vdash_m \lambda x.R : (F, M - \mathsf{Mk}(C), C \to \tau) \triangleright e$. Notice that $\mathsf{Mk}(C) \le \mathsf{Mk}(\Upsilon[x \mapsto C]) \le M$ by Lemma 5.2, so it makes sense to use $M - \mathsf{Mk}(C)$. Denoting $\tau = C_1 \to \ldots \to C_s \to o$, we also need to know that $(M - \mathsf{Mk}(C) + \mathsf{Mk}(C) + \sum_{i=1}^s \mathsf{Mk}(C_i))(0) = 1$ (this is required by the definition of a type triple), but it follows immediately from $(M + \sum_{i=1}^s \mathsf{Mk}(C_i))(0) = 1$.

To this type judgment, and to $\Delta_i \vdash_m S : \hat\sigma_i \triangleright d_i$ for $i \in I$, we want to apply the (@) rule. By the definition of a type judgment, the type triples $\hat\sigma_i$, hence also the sets F_i and M_i, are m-bounded. Recalling that $ord(S) = ord(x) = m$ we have that the type $\{\!\{(F_i{\restriction}_{\le ord(S)}, M_i{\restriction}_{\le ord(S)}, \sigma_i) \mid i \in I\}\!\} \to \tau$ that we have to derive for $\lambda x.R$ is indeed $C \to \tau$, and the side condition $ord(S) \le m$ is satisfied. The resulting marker multiset is $(M - \mathsf{Mk}(C)) + \sum_{i\in I} M_i = M$, and the resulting type environment is $\Upsilon \sqcup \bigsqcup_{i\in I}\Delta_i = \Gamma$. Notice that the sets $F_i{\restriction}_{>ord(S)}$ are empty, and that $M(k) = 0$ for all $(k,a) \in F$ by the definition of a type triple ($\hat\tau = (F, M, \tau)$ is a type triple), and hence $(F, c) \in Comp_m(M; (F, e), ((F_i{\restriction}_{>ord(S)}, d_i))_{i\in I})$ by Lemma 6.6. The condition $\{(k,a) \in F \mid M(k) = 0\} \subseteq F$ is clearly satisfied. Thus the (@) rule can be applied, and it derives $\Gamma \vdash_m P : \hat\tau \triangleright c$.

It remains to consider the general situation: the redex involved in the beta-reduction $P \to_{\beta(m)} Q$ is located somewhere deeper in P. Then the proof is by a trivial induction on the depth of this redex. Formally, we have several cases depending on the shape of P, but let us consider only a representative example: suppose that $P = T\,U$ and $Q = T\,V$ with $U \to_{\beta(m)} V$. In the derivation of $\Gamma \vdash_m Q : \hat{\tau} \triangleright c$ we apply the induction assumption to those premisses of the final (@) rule that concern the subterm V, and we obtain type judgments in which V is replaced by U. We can apply the (@) rule to them, and to the premiss talking about T, and derive $\Gamma \vdash_m P : \hat{\tau} \triangleright c$. □

6.3 Proof of Lemma 6.4

Recall that in Lemma 6.4 we are given a type derivation of order $m-1$, and we want to convert it into a type derivation of order m, without decreasing the flag counter too much. The order of the derivation can be raised without any problem, we only need to additionally place $|\Sigma|$ markers of order m in some leaves of the derivation. We notice, however, that in the original derivation the flag counter computed the number of order-m flags, while in the new derivation it computes the number of order-$(m+1)$ flags. We thus have to ensure that many order-$(m+1)$ flags are created in the new derivation. To this end, we appropriately choose where the order-m markers are placed. Let us now give more details.

For the rest of the subsection fix an order $m \geq 1$. We shall see derivations as trees. A *derivation tree* is a finite tree with nodes labeled by type judgments, such that for every node, the label of this node can be obtained by applying some rule of the type system to labels of children of this node. We consider derivation trees only for type judgments of order $m-1$ (that is, only the derivation that we receive as the input to the lemma is seen as a tree, not the one that we produce). For a derivation tree t and for its node v, by t_v we denote the subtree of t starting at v, and by c_v we denote the flag counter being part of the type judgment written in v.

The proof is done in two steps: first, we label the derivation tree by some additional flags and markers, and then, based on such a labeling, we construct a derivation of order m. For $B \subseteq \Sigma$, and for a derivation tree t, a *B-labeling of t* assigns some number of order-m markers to every leaf of t, and for every $a \in B$, some number of (m,a)-flags to every node of t. In the sequel, we simply talk about assigning markers and a-flags, having implicitly in mind that they are of order m. A B-labeling ρ of t is *consistent*, if:

- for every node v of t having children $v_1, \ldots, v_k$, and for every $a \in B$, ρ assigns at most $c_v(a) - c_{v_1}(a) - \cdots - c_{v_k}(a)$ a-flags to v, and
- in every subtree of t in which ρ assigns no markers, ρ assigns at most one flag.

Observe that our type system ensures that the number $c_v(a) - c_{v_1}(a) - \cdots - c_{v_k}(a)$ appearing above is always nonnegative: the flag counter in every node is not smaller than the sum of flag counters coming from the premisses.

We start by showing how to construct B-labelings in the case when B consists of a single letter.

Lemma 6.7. *Let t be a derivation tree with root r, and let $a \in \Sigma$. Then there exists a consistent $\{a\}$-labeling ρ_a of t that assigns in total exactly one marker and at least $\log_2 c_r(a)$ a-flags.*

Proof: Induction on the size of t. If t consists of a single node, then to this node we assign one marker, and $c_r(a)$ a-flags. Such a labeling is consistent, and we have $c_r(a) \geq \log_2 c_r(a)$.

Suppose now that r has some children $v_1, \ldots, v_k$ with $k \geq 1$. Fix some s for which $c_{v_s}(a)$ is maximal, that is, such that $c_{v_s}(a) \geq c_{v_i}(a)$ for all $i \in \{1, \ldots, k\}$. We apply the induction assumption to the subtree t_{v_s}; it gives us a consistent $\{a\}$-labeling of this subtree, which assigns in total exactly one marker and at

least $\log_2 c_{v_s}(a)$ a-flags. Moreover, for every $i \in \{1,\ldots,k\} \setminus \{s\}$ such that $c_{v_i}(a) > 0$, we choose some node w_i in the subtree t_{v_i} so that $c_{w_i}(a) > 0$ but $c_u(a) = 0$ for every child u of w_i (clearly such a node exists), and we assign an a-flag to the chosen node w_i. Finally, we denote $l = c_r(a) - c_{v_1}(a) - \cdots - c_{v_k}(a)$, and to the root of t we assign l a-flags. It should be clear that the obtained $\{a\}$-labeling is consistent.

It remains to observe that the number f of assigned a-flags is at least $\log_2 c_r(a)$. In the degenerate case of $c_{v_s}(a) = 0$ we have $c_{v_i}(a) = 0$ for all $i \in \{1,\ldots,k\}$, and thus $f = l = c_r(a) \geq \log_2 c_r(a)$. Suppose now that $c_{v_s}(a) > 0$, and denote $l' = |\{i \in \{1,\ldots,k\} \mid c_{v_i}(a) > 0\}|$. Then by construction we have $f \geq l + (l'-1) + \log_2 c_{v_s}(a)$. Recalling that $c_{v_s}(a) > 0$ and $c_{v_s}(a) \geq c_{v_i}(a)$ for all $i \in \{1,\ldots,k\}$, we obtain

$$\begin{aligned} f \geq l + l' - 1 + \log_2 c_{v_s}(a) &\geq \log_2(l + l') + \log_2 c_{v_s}(a) \\ &= \log_2((l + l') \cdot c_{v_s}(a)) \\ &\geq \log_2(l + l' \cdot c_{v_s}(a)) \\ &\geq \log_2(l + c_{v_1}(a) + \cdots + c_{v_k}(a)) = \log_2 c_r(a)\,. \end{aligned}$$ □

Having labelings concerning particular letters, we now show how to merge them into a single labeling concerning all letters.

Lemma 6.8. *Let t be a derivation tree with root r. Then there exists a consistent Σ-labeling of t that assigns exactly $|\Sigma|$ markers and at least $\left\lfloor \frac{1}{|\Sigma|} \log_2 c_r(a) \right\rfloor$ a-flags, for every $a \in \Sigma$.*

Proof: We start by applying Lemma 6.7 for every letter $a \in \Sigma$, which results in a consistent a-labeling ρ_a of t. Based on these labelings we construct the resulting labeling ρ. For every node v of t, if k among labelings ρ_a assign a marker to v, then in ρ we assign k markers to v. This assigns $|\Sigma|$ markers in total. For every node v of t such that ρ assigns some markers in t_v, and for every $a \in \Sigma$, if ρ_a assigns k a-flags to v, then in ρ we also assign k a-flags to v.

Let now V be the set of all nodes v such that ρ assigns no markers in t_v, but it assigns some markers in the subtree starting in the parent of v. In subtrees starting in $v \in V$ there may be plenty of flags assigned by the labelings ρ_a, and we have not yet taken these flags to ρ. We do this now, using the following algorithm: we repeat the *big step* as long as it gives something new. In a big step, we execute the *small step* for every letter $a \in \Sigma$. In a small step concerning some letter a, we choose some $v \in V$ such that ρ_a assigns an a-flag to some node w in the subtree t_v, but ρ does not assign any flag in this subtree yet; if such nodes v, w exist, then in ρ we assign an a-flag to w.

We notice that ρ assigns in every node of t at most as many a-flags as ρ_a did. Moreover, in every subtree starting in a node of V (and thus in every subtree of t in which ρ assigns no markers), ρ assigns at most one flag. This means that ρ is consistent.

It remains to observe that the number of assigned flags is large enough. Fix some $a \in \Sigma$. Let f_a be the total number of a-flags assigned by ρ_a; by Lemma 6.7 we have $f_a \geq \log_2 c_r(a)$. These flags are of two kinds: we have $f_a = g_a + h_a$, where g_a is the total number of a-flags assigned by ρ_a to nodes w such that ρ assigns some marker in t_w, and h_a is the number of a-flags assigned by ρ_a to remaining nodes. The g_a flags of the first kind are simply copied to ρ. Let us now look closer on the flags of the second kind. Every node w such that ρ assigns no markers in t_w, belongs to t_v for some $v \in V$. Moreover, for every $v \in V$, ρ_a assigns at most one a-flag in t_v. It follows that h_a equals the number of nodes $v \in V$ such that ρ_a assigns some a-flag in t_v. In every small step we assign a flag in the subtree t_v for at most one node $v \in V$, and

thus in every big step we assign a flag in the subtrees t_v for at most $|\Sigma|$ nodes $v \in V$. This means that during the first $\left\lfloor \frac{h_a}{|\Sigma|} \right\rfloor$ big steps there still exists a node $v \in V$ such that ρ_a assigns an a-flag in t_v, but ρ does not assign any flag in this subtree yet, and thus a new a-flag is assigned by ρ. In consequence, the number of a-flags assigned by ρ is at least $g_a + \left\lfloor \frac{h_a}{|\Sigma|} \right\rfloor \geq \left\lfloor \frac{f_a}{|\Sigma|} \right\rfloor \geq \left\lfloor \frac{1}{|\Sigma|} \log_2 c_r(a) \right\rfloor$. □

Next, we show how to raise the order of a type derivation based on a consistent labeling. In this part, it is convenient to assume that the labeling is maximal, in the following sense: a consistent Σ-labeling ρ of a derivation tree t is called *maximal* if for every $a \in \Sigma$ and for every node v of t having children $v_1, \ldots, v_k$, if ρ assigns some marker in t_v, then ρ assigns exactly $c_v(a) - c_{v_1}(a) - \cdots - c_{v_k}(a)$ a-flags to v. Notice that in such nodes this is the maximal number of flags allowed by the first point in the definition of consistency. We cannot require anything similar from nodes v such that no marker is assigned in t_v, as the number of flags in those nodes is strongly restricted by the second point of the definition.

We now define functions New_{M} and New_{Fc}: we say that $New_{\mathsf{M}}(M, \mu) = M'$ and $New_{\mathsf{Fc}}(F, \mu, f) = (F', c')$ if

- $M'(k) = M(k)$ for $k \neq m$, and $M'(m) = \mu$,
- if $\mu > 0$, then $F' = F$ and $c' = f$, and
- if $\mu = 0$, then $F' = F \cup \{(m, a) \mid f(a) > 0\}$ and $c' = \mathbf{0}$.

The intended meaning is that if M and F are a marker multiset and a flag set derived in some node v of a derivation tree, and in t_v a labeling assigns μ markers and $f(a)$ a-flags for every $a \in \Sigma$, then in the new derivation that we construct, we use M' as the marker multiset, F' as the flag set, and c' as the flag counter. Notice that the previous value of the flag counter is not taken into account. We now have a lemma saying that the *Comp* predicate remains satisfied after applying the transformation.

Lemma 6.9. *Suppose that $(F, c) \in Comp_{m-1}(M; ((F_i, c_i))_{i \in I})$, where $F \in \mathcal{F}_{m-1}$. For $i \in I$ let $\mu_i \in \mathbb{N}$ and $f_i \colon \Sigma \to \mathbb{N}$. Suppose also that $\mu \geq \sum_{i \in I} \mu_i$, and $f \leq \sum_{i \in I} f_i + c - \sum_{i \in I} c_i$, and if $\mu > 0$ then $f = \sum_{i \in I} f_i + c - \sum_{i \in I} c_i$. Finally, for every $i \in I$ suppose that if $\mu_i = 0$ then $f_i(a) \leq 1$ for all $a \in \Sigma$, and that either*

- *$F_i \in \mathcal{F}_{m-1}$, or*
- *$f_i = 0$ and $F_i = \{(1, a)\}$ for some $a \in \Sigma$.*

In such a situation, $New_{\mathsf{Fc}}(F, \mu, f) \in Comp_m(New_{\mathsf{M}}(M, \mu); (New_{\mathsf{Fc}}(F_i, \mu_i, f_i))_{i \in I})$.

Proof: Denote $M' = New_{\mathsf{M}}(M, \mu)$, $(F', c') = New_{\mathsf{Fc}}(F, \mu, f)$, and $(F'_i, c'_i) = New_{\mathsf{Fc}}(F_i, \mu_i, f_i)$ for $i \in I$. We consider the numbers $f_{k,a}$ and $f'_{k,a}$ appearing in the definition of $Comp_{m-1}(M; ((F_i, c_i))_{i \in I})$. We also consider analogous numbers defined by the predicate $Comp_m(M'; ((F'_i, c'_i))_{i \in I})$, and we call them $g_{k,a}$ and $g'_{k,a}$. Since $M' \restriction_{\leq m-1} = M \restriction_{\leq m-1}$ and $F'_i \restriction_{\leq m-1} = F_i \restriction_{\leq m-1}$ for $i \in I$, for every $a \in \Sigma$ we clearly have that $g_{k,a} = f_{k,a}$ for $k \leq m-1$, and $g'_{k,a} = f'_{k,a}$ for $k \leq m$. Moreover, $g_{m+1,a} = g'_{m+1,a} + \sum_{i \in I} |F'_i \cap \{(m+1, a)\}| = g'_{m+1,a}$ since the sets F'_i are m-bounded (notice that $F_i = \{(1, a)\}$ does not need to be $(m-1)$-bounded, but surely is m-bounded).

Let us now see that for all $i \in I$ and $a \in \Sigma$ it is the case that

$$|F_i \cap \{(m, a)\}| + f_i(a) = |F'_i \cap \{(m, a)\}| + c'_i(a)\,. \tag{1}$$

Indeed:

- if $\mu_i > 0$, then $F'_i = F_i$ and $c'_i = f_i$;
- if $\mu_i = 0$ and $f_i(a) = 0$, then $c'_i(a) = 0$ and $F'_i = F_i$;

- if $\mu_i = 0$ and $f_i(a) > 0$, then $c'_i(a) = 0$, and $(m,a) \notin F_i$ (since $F_i \in \mathcal{F}_{m-1}$), and F'_i contains (m,a) (by the definition of F'_i), and $f_i(a) \leq 1$ (by assumption), which gives Equality (1).

Using Equality (1) we observe that for every $a \in \Sigma$,

$$\begin{aligned}\sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) &= \sum_{i\in I} f_i(a) + f_{m,a}\\ &= f'_{m,a} + \sum_{i\in I} |F_i \cap \{(m,a)\}| + \sum_{i\in I} f_i(a)\\ &= g'_{m,a} + \sum_{i\in I} |F'_i \cap \{(m,a)\}| + \sum_{i\in I} c'_i(a)\\ &= g_{m,a} + \sum_{i\in I} c'_i(a)\,. \end{aligned} \tag{2}$$

In order to obtain the conclusion of the lemma, we need to prove two facts: that $F' \subseteq \{(k,a) \mid g_{k,a} > 0\}$, and that $c'(a) = g_{m+1,a} + \sum_{i\in I} c'_i(a)$ for all $a \in \Sigma$. We first concentrate on the part $F' \subseteq \{(k,a) \mid g_{k,a} > 0\}$. By assumption we have that $F \subseteq \{(k,a) \mid f_{k,a} > 0\}$, and thus also $F \subseteq \{(k,a) \mid g_{k,a} > 0\}$ since F is $(m-1)$-bounded, and since $g_{k,a} = f_{k,a}$ for $k \leq m-1$. When $\mu > 0$, we have $F' = F$, and we are done. Suppose thus that $\mu = 0$. Then F' contains also elements (m,a) for all $a \in \Sigma$ such that $f(a) > 0$. Concentrate on one such a. By assumption and by Equality (2) we obtain that

$$0 < f(a) \leq \sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) = g_{m,a} + \sum_{i\in I} c'_i(a)\,.$$

Since $0 = \mu \geq \sum_{i\in I} \mu_i$, for every $i \in I$ we have $\mu_i = 0 = c'_i(a)$, and thus $g_{m,a} > 0$ by the above inequality. We thus have $(m,a) \in \{(k,a) \mid g_{k,a} > 0\}$, as required.

Next, we fix some $a \in \Sigma$, and we prove that $c'(a) = g'_{m+1,a} + \sum_{i\in I} c'_i(a)$, which is what we need since $g_{m+1,a} = g'_{m+1,a}$. Suppose first that $\mu = 0$. Then $c'(a) = 0$ and $c'_i(a) = 0$ for all $i \in I$, since $\mu = 0$ implies $\mu_i = 0$. We also have $M'(m) = \mu = 0$, and thus $g'_{m+1,a} = 0$, which gives the thesis. Next, suppose that $\mu > 0$. In such a case, using Equality (2), we obtain that

$$c'(a) = f(a) = \sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) = g_{m,a} + \sum_{i\in I} c'_i(a) = g'_{m+1,a} + \sum_{i\in I} c'_i(a)\,. \qquad \square$$

In the next lemma we show how a labeling is used to construct a derivation of order m.

Lemma 6.10. *Let t be a derivation tree deriving $\Gamma \vdash_{m-1} R : (F, M, \tau) \triangleright c$ such that $ord(R) \leq m$, and $\Gamma(x) \neq \mathbf{0}$ only for variables x of order at most $m-1$. Let also ρ be a maximal consistent Σ-labeling of t, which assigns (in total) $\mu \leq |\Sigma|$ markers and $f(a)$ a-flags, for every $a \in \Sigma$. Then we can derive $\Gamma \vdash_m R : (F', M', \tau) \triangleright c'$, where $M' = New_{\mathsf{M}}(M, \mu)$ and $(F', c') = New_{\mathsf{Fc}}(F, \mu, f)$.*

Proof: Denote $\hat{\tau} = (F, M, \tau)$ and $\hat{\tau}' = (F', M', \tau)$. We first prove that $\hat{\tau}'$ is indeed an m-bounded type triple. By assumption $\hat{\tau}$ is an $(m-1)$-bounded type triple, so $M \in \mathcal{M}_{m-1}$. Since M' differs from M only on order m, and $M'(m) = \mu \leq |\Sigma|$, we obtain that $M' \in \mathcal{M}_m$ (recall that $m \geq 1$). We also have $F \in \mathcal{F}_{m-1} \subseteq \mathcal{F}_m$. The set F' differs from F only when $\mu = 0$, and then it additionally contains those pairs (m,a) for which $f(a) > 0$. By consistency of ρ we know that if $\mu = 0$ (i.e., if ρ assigns no markers) then

$\sum_{a\in\Sigma} f(a) \leq 1$. Thus $(m,a),(m,b) \in F'$ implies $a = b$, which establishes that $F' \in \mathcal{F}_m$. We also need to know that $M'(k) = 0$ for all $(k,a) \in F'$. For $k \leq m-1$ this is the case because $M'{\restriction}_{\leq m-1} = M$ and $F'{\restriction}_{\leq m-1} = F$, and by definition of F' we have $(m,a) \in F'$ only when $M'(m) = \mu = 0$. Additionally, we need to know that $M'(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1$, where $\tau = C_1 \to \ldots \to C_s \to o$; this is the case because $M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1$ and $M'(0) = M(0)$ due to $m \geq 1$.

The rest of the proof is by induction on the size of t. We have several cases depending on the shape of R.

Suppose first that $R = x$ is a variable. Then the (VAR) rule used in the only node of t ensures that $c = \mathbf{0}$ and that $\Gamma = \varepsilon[x \mapsto \{\!\{(F, M{\restriction}_{\leq ord(x)}, \tau)\}\!\}]$. By assumptions of the lemma $ord(x) \leq m-1$, so $M'{\restriction}_{\leq ord(x)} = M{\restriction}_{\leq ord(x)}$. Moreover $F' = F$ and $c' = \mathbf{0}$ since $f \leq c = \mathbf{0}$ by consistency of ρ. Thus the (VAR) rule can equally well derive $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$ (notice that $c' = \mathbf{0}$).

Next, suppose that $R = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$. Then the root of t has exactly one child v, labeled by the premiss of the (ND) rule, $\Gamma \vdash_{m-1} P_i : \hat{\tau} \triangleright c$ for some $i \in \{1,\ldots,r\}$. Since the flag counter is the same as in the root, ρ assigns no flags to the root of t (by consistency of ρ). Thus the induction assumption applied to t_v gives us a derivation of $\Gamma \vdash_m P_i : \hat{\tau}' \triangleright c'$. Applying back the (ND) rule we derive $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$.

Suppose now that $R = \lambda x.P$. Then the root of t has exactly one child v, labeled by the premiss of the (λ) rule, $\Gamma[x \mapsto C'] \vdash_{m-1} P : (F, M_\lambda, \tau_\lambda) \triangleright c$, where $\tau = C \to \tau_\lambda$, and $M = M_\lambda - \mathsf{Mk}(C)$, and $C' \sqsubseteq C$. As in the previous case, no flags are assigned to the root of t. Because $ord(R) \leq m$, we have $ord(P) \leq m$ and $ord(x) \leq m-1$, so assumptions of the lemma are satisfied for t_v; the induction assumption gives us a derivation of $\Gamma[x \mapsto C'] \vdash_m P : (F', M'_\lambda, \tau_\lambda) \triangleright c'$, where $M'_\lambda = New_{\mathsf{M}}(M_\lambda, \mu)$. The triple container C is $ord(x)$-bounded, thus since $ord(x) \leq m-1$ we have $M'(m) = \mu = M'_\lambda(m) = M'_\lambda(m) - \mathsf{Mk}(C)(m)$, and hence $M' = M'_\lambda - \mathsf{Mk}(C)$. Thus after applying back the (λ) rule we obtain a derivation of $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$.

Next, suppose that $R = a\langle P\rangle$, where $a \neq \mathsf{nd}$. The root of t has exactly one child v, labeled by the premiss of the (CON1) rule, $\Gamma \vdash_{m-1} P : (F_1, M, o) \triangleright c_1$. Denote by $f_1(a)$ the number of a-flags assigned by ρ in t_v, for every $a \in \Sigma$. By the induction assumption, we can derive $\Gamma \vdash_m P : (F'_1, M', o) \triangleright c'_1$, where $(F'_1, c'_1) = New_{\mathsf{Fc}}(F_1, \mu, f_1)$. The (CON1) rule ensures that $\tau = o$, and that $(F, c) \in Comp_{m-1}(M; (\{(1,a)\}, \mathbf{0}), (F_1, c_1))$. We want to apply Lemma 6.9 with $I = \{0,1\}$, $\mu_0 = 0$, $f_0 = \mathbf{0}$, $F_0 = \{(1,a)\}$, $c_0 = \mathbf{0}$, $\mu_1 = \mu$; let us check its assumptions. Clearly $\mu \geq \sum_{i\in I} \mu_i$. By consistency of ρ, and because $f_0 = c_0 = \mathbf{0}$, we have that $f \leq \sum_{i\in I} f_i + c - \sum_{i\in I} c_i$, and that for every $i \in I$, if $\mu_i = 0$ then $f_i(a) \leq 1$ for all $a \in \Sigma$. By maximality of ρ we have that if $\mu > 0$ then $f = \sum_{i\in I} f_i + c - \sum_{i\in I} c_i$. Moreover, $New_{\mathsf{Fc}}(F_0, \mu_0, f_0) = (\{(1,a)\}, \mathbf{0})$. Thus, by Lemma 6.9 we obtain that $(F', c') \in Comp_m(M'; (\{(1,a)\}, \mathbf{0}), (F'_1, c'_1))$. Applying back the (CON1) rule we can derive $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$.

The case $R = a\langle\rangle$ with $a \neq \mathsf{nd}$ is similar. The root of t is a leaf using the (CON0) rule, and $\Gamma = \varepsilon$, and $\tau = o$, and $(F, c) \in Comp_{m-1}(M; (\{(1,a)\}, \mathbf{0}))$. We apply Lemma 6.9 with $I = \{0\}$, $\mu_0 = 0$, $f_0 = \mathbf{0}$, $F_0 = \{(1,a)\}$, $c_0 = \mathbf{0}$; its assumptions are easily satisfied. This lemma implies that $(F', c') \in Comp_m(M'; (\{(1,a)\}, \mathbf{0}))$, so applying back the (CON0) rule we can derive $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$.

Finally, suppose that $R = P\,Q$. Let $\Gamma_0 \vdash_{m-1} P : (F_0, M_0, C \to \tau) \triangleright c_0$ and $\Gamma_i \vdash_{m-1} Q : (F_i, M_i, \tau_i) \triangleright c_i$ for each $i \in I$ be the premisses of the (@) rule used in the root of t, where $C = \{\!\{(F_i{\restriction}_{\leq ord(Q)}, M_i{\restriction}_{\leq ord(Q)}, \tau_i) \mid i \in I\}\!\}$, and where without loss of generality we assume that $0 \notin I$. Denote the children of the root of t having these type judgments as labels by v_i for $i \in \{0\} \cup I$, respectively. For $i \in \{0\} \cup I$ denote by μ_i the number of markers assigned by ρ in t_{v_i}, and by $f_i(a)$ the number of a-flags assigned by ρ in t_{v_i}, for every $a \in \Sigma$. The (@) rule ensures that $\Gamma = \bigsqcup_{i\in\{0\}\cup I} \Gamma_i$ and $ord(Q) \leq m-1$, and by homogeneity of the sort of P we obtain that $ord(P) \leq ord(Q) + 1 \leq m$. This allows us to apply the induction assumption, which gives us derivations of $\Gamma_0 \vdash_m P : (F'_0, M'_0, C \to \tau) \triangleright c'_0$ and

$\Gamma_i \vdash_m Q : (F'_i, M'_i, \tau_i) \triangleright c'_i$ for each $i \in I$, where $M'_i = New_{\mathsf{M}}(M_i, \mu_i)$ and $(F'_i, c'_i) = New_{\mathsf{Fc}}(F_i, \mu_i, f_i)$ for all $i \in \{0\} \cup I$. To these type judgments we would like to apply the (@) rule, but we need to check its conditions.

- Since $F'_i\restriction_{\leq m-1} = F_i\restriction_{\leq m-1}$ and $M'_i\restriction_{\leq m-1} = M_i\restriction_{\leq m-1}$ for all $i \in I$, and $ord(Q) \leq m-1$, we have that $\bar{C} = \{\!\{(F'_i\restriction_{\leq ord(Q)}, M'_i\restriction_{\leq ord(Q)}, \tau_i) \mid i \in I\}\!\}$.
- Notice that $M = \sum_{i\in\{0\}\cup I} M_i$, due to the original use of the (@) rule, and that $\mu = \sum_{i\in\{0\}\cup I} \mu_i$, by definition. Recalling that M'_i and M' are obtained from M_i and M by inserting μ_i and μ at order m, we obtain that $M' = \sum_{i\in\{0\}\cup I} M'_i$.
- The original use of the (@) rule gives us that $(F, c) \in Comp_{m-1}(M; (F_0, c_0), ((F_i\restriction_{>ord(Q)}, c_i))_{i\in I})$. Assumptions of Lemma 6.9 are satisfied: $\mu \geq \sum_{i\in\{0\}\cup I} \mu_i$ by definition; $f \leq \sum_{i\in\{0\}\cup I} f_i + c - \sum_{i\in\{0\}\cup I} c_i$ by consistency of ρ; for every $i \in \{0\} \cup I$, if $\mu_i = 0$ then $f_i(a) \leq 1$ for all $a \in \Sigma$, again by consistency of ρ; finally, if $\mu > 0$ then $f = \sum_{i\in\{0\}\cup I} f_i + c - \sum_{i\in\{0\}\cup I} c_i$ by maximality of ρ. Moreover, since the New_{Fc} function modifies only order m, and $ord(Q) \leq m-1$, we have $New_{\mathsf{Fc}}(F_i\restriction_{>ord(Q)}, \mu_i, f_i) = (F'_i\restriction_{>ord(Q)}, c'_i)$. Thus by Lemma 6.9 we obtain that $(F', c') \in Comp_m(M'; (F'_0, c'_0), ((F'_i\restriction_{>ord(Q)}, c'_i))_{i\in I})$.
- We also need to prove that $\{(k, a) \in F'_0 \mid M'(k) = 0\} \subseteq F'$. We know that $\{(k, a) \in F_0 \mid M(k) = 0\} \subseteq F$, and since F'_0, M', F' differ from F_0, M, F only on order m, we only need to check for all $a \in \Sigma$ that if $M'(m) = 0$ and $(m, a) \in F'_0$ then also $(m, a) \in F'$. This is clear: $(m, a) \in F'_0$ may only happen when $f_0(a) > 0$, but then $f(a) \geq f_0(a) > 0$, which in the case of $\mu = M'(m) = 0$ implies that $(m, a) \in F'$.

All this allows us to apply back the (@) rule, and derive $\Gamma \vdash_m R : \hat{\tau}' \triangleright c'$. □

Proof of Lemma 6.4: Consider a derivation tree t that derives $\varepsilon \vdash_{m-1} P : \hat{\rho}^{all}_{m-1} \triangleright c$. Using Lemma 6.8 we construct a consistent Σ-labeling ρ of t that assigns exactly $|\Sigma|$ markers and at least $\left\lfloor \frac{1}{|\Sigma|} \log_2 c(a) \right\rfloor$ a-flags, for every $a \in \Sigma$. Without loss of generality we can assume that ρ is maximal: if not, we simply add more flags in some nodes, as required by the maximality condition.[(iii)] Then Lemma 6.10 gives us a derivation of $\varepsilon \vdash_m P : \hat{\rho}^{all}_m \triangleright c'$ for some c' such that $c'(a) \geq \left\lfloor \frac{1}{|\Sigma|} \log_2 c(a) \right\rfloor$ for all $a \in \Sigma$, as required. □

7 Soundness

In this section we prove the right-to-left implication of Theorem 3.2. As a side effect, we also obtain a proof of Lemma 5.5. We, basically, need to reverse the proof from the previous section. We give the following three lemmata (corresponding to Lemmata 6.2-6.4; Lemma 6.1 is used also in this section, without any change).

Lemma 7.1. *Suppose that $P \to_{\beta(m)} Q$, where $m \geq 0$. If we can derive $\Gamma \vdash_m P : \hat{\tau} \triangleright c$, then we can also derive $\Gamma' \vdash_m Q : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$. Moreover, if the original derivation was wild, then the resulting one is also wild.*

Lemma 7.2. *Let P be a closed lambda-term of complexity at most m. If we can derive $\varepsilon \vdash_m P : \hat{\rho}^{all}_m \triangleright c$, where $m \geq 1$, then we can also derive $\varepsilon \vdash_{m-1} P : \hat{\rho}^{all}_{m-1} \triangleright c'$ for some $c' \geq c$. Moreover, if the original derivation was wild, then the resulting one is also wild.*

[(iii)] We notice that usually the labeling constructed by Lemma 6.8 is not maximal.

Lemma 7.3. *Suppose that we can derive $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$, where P is a lambda-term of complexity 0. Then there exists a word $T \in \mathcal{L}(P)$ such that for every $a \in \Sigma$, the number of occurrences of a in T is $c(a)$.*

Let us now see how the right-to-left implication of Theorem 3.2 follows from these lemmata. Thus, take a closed lambda-term P of sort o and of complexity at most $m+1$, and a set $A \subseteq \Sigma$, and suppose that for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_m$ for some c_m such that $c_m(a) \geq n$ for all $a \in A$. We want to prove that for every $n \in \mathbb{N}$ there is a tree $T \in \mathcal{L}(P)$ in which every letter $a \in A$ appears at least n times. To this end, take some $n \in \mathbb{N}$, and the type judgment corresponding to this n. Using Lemma 4.1 we can find a finite lambda-term $P' \preccurlyeq P$ for which we can also derive $\varepsilon \vdash_m P' : \hat{\rho}_m^{all} \triangleright c_m$. Then, we apply Lemma 6.1 to P', obtaining lambda-terms $Q_{m+1}, Q_m, \ldots, Q_0$ such that, for every $k \in \{0, \ldots, m\}$, the complexity of Q_k is at most k, and $Q_{k+1} \to^*_{\beta(k)} Q_k$, and $Q_{m+1} = P'$. Next, consecutively for $k = m, m-1, \ldots, 0$ we perform two steps. First, we repeatedly apply Lemma 7.1 to every beta-reduction (of order k) between Q_{k+1} and Q_k, obtaining a derivation of $\varepsilon \vdash_k Q_k : \hat{\rho}_k^{all} \triangleright c_k$. Then, if $k \geq 1$, we apply Lemma 7.2, obtaining a derivation of $\varepsilon \vdash_{k-1} Q_k : \hat{\rho}_{k-1}^{all} \triangleright c_{k-1}$ for some $c_{k-1} \geq c_k$. We end up with a derivation of $\varepsilon \vdash_0 Q_0 : \hat{\rho}_0^{all} \triangleright c_0$, where $c_0(a) \geq c_m(a) \geq n$ for all $a \in A$. By Lemma 7.3 we can find a word $T \in \mathcal{L}(Q_0) = \mathcal{L}(BT(P'))$ such that for every $a \in A$, the number of occurrences of a in T as at least n. Due to Lemma 4.2, we also have $T \in \mathcal{L}(BT(P))$, as needed.

We also obtain a proof of Lemma 5.5. Indeed, suppose that we have a wild derivation of $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_m$ for a closed lambda-term P of sort o and of complexity at most $m+1$. Lemma 4.1 implies that we can find a finite lambda-term $P' \preccurlyeq P$ for which we can also derive $\varepsilon \vdash_m P' : \hat{\rho}_m^{all} \triangleright c_m$. By inspecting the proof of this lemma we notice that if the derivation for P was wild, then the derivation for P' is also wild (because this is essentially the same derivation). Then, by applying the same arguments as above, we obtain a derivation of $\varepsilon \vdash_0 Q_0 : \hat{\rho}_0^{all} \triangleright c_0$ for some lambda-term Q_0 of complexity 0. Moreover, this derivation is wild, since Lemmata 7.1 and 7.2 preserve wildness. On the other hand, a lambda-term of complexity 0 does not contain any applications, so our derivation does not use the (@) rule at all, and hence it cannot be wild. This is a contradiction implying that there could not exist a wild derivation of $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_m$.

In the remaining part of this section we prove the three lemmata.

7.1 Proof of Lemma 7.1

The overall idea of the proof is very simple: when $P = (\lambda x.R)\, S$ and $Q = R[S/x]$, we perform a surgery on the derivation concerning P and we obtain a derivation concerning Q. Namely, whenever the subderivation concerning R uses the (VAR) rule for the variable x, we should insert there a subderivation that derives the same type triple for S. We need to notice that every unbalanced type triple derived for S is used for exactly one occurrence of x in the derivation concerning R. Balanced type triples may be used many times, or not used at all, but we can see that duplicating or removing the corresponding derivations for S is not problematic; in particular it does not change the flag counter, as shown in Lemma 5.4.

We start the proof by showing in Lemma 7.4 how type derivations may be composed during a substitution. This lemma can be seen as a converse of Lemma 6.5.

Lemma 7.4. *Suppose that we can derive $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I$, and $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$, where $\Upsilon^x = \Upsilon[x \mapsto \{\!\{\hat{\sigma}_i \mid i \in I\}\!\}]$ for a variable x of order m and of the same sort as S, and $\Gamma = \Upsilon \sqcup \bigsqcup_{i \in I} \Delta_i$ is a type environment. Then we can also derive $\Gamma' \vdash_m R[S/x] : \hat{\tau} \triangleright c$ for $c = e + \sum_{i \in I} d_i$ and for some $\Gamma' \sqsubseteq \Gamma$. Moreover, if some of the original derivations are wild, then the resulting derivation is also wild.*

Proof: The proof is by induction on the structure of some fixed derivation of $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$.

One possibility is that x is not free in R. In such a situation $R[S/x] = R$ and $\Upsilon^x(x) = \mathbf{0}$, so $I = \emptyset$, and $\Gamma = \Upsilon = \Upsilon^x$, and $c = e$, thus we can derive $\Gamma \vdash_m R[S/x] : \hat{\tau} \triangleright c$ by assumption.

In the sequel we assume that x is free in R; in particular, R is not of the form $a\langle\rangle$, nor R is a variable other than x. We analyze the shape of R.

Suppose first that $R = x$ is a variable. Then $R[S/x] = S$, and the derivation for R consists of a single use of the (VAR) rule, thus $e = \mathbf{0}$ and $\Upsilon^x = \varepsilon[x \mapsto \{\!|\hat{\tau}|\!\}]$ (since $ord(x) = m$, no new markers could be added). It means that $\Upsilon = \varepsilon$, and $\{\!|\hat{\tau}|\!\} = \{\!|\hat{\sigma}_i \mid i \in I|\!\}$. We have two subcases.

- Suppose first that $\hat{\tau}$ is unbalanced. Then necessarily $|I| = 1$, say $I = \{1\}$. It follows that $\Gamma = \Delta_1$, and $c = d_1$, so we can derive $\Gamma \vdash_m R[S/x] : \hat{\tau} \triangleright c$ by assumption.
- The situation of an unbalanced $\hat{\tau}$ is slightly different. We only know that $|I| \geq 1$ and $\hat{\tau} = \hat{\sigma}_i$ for all $i \in I$. Then from Lemma 5.2 we obtain that $\mathsf{Mk}(\Delta_i) \leq \mathsf{Mk}(\hat{\sigma}_i) = \mathbf{0}$ for all $i \in I$, that is, that all type triples in all Δ_i are balanced. In consequence, $\Delta_i \sqsubseteq \Gamma$ (due to $\Gamma = \bigsqcup_{i \in I} \Delta_i$). Similarly, from Lemma 5.4 we obtain that $d_i = \mathbf{0}$ for all $i \in I$, so $c = d_i$. Thus as the resulting derivation we can take $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for any $i \in I$.

Next, suppose that $R = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$. Then the derivation for R ends with the (ND) rule, whose premiss is $\Upsilon^x \vdash_m P_k : \hat{\tau} \triangleright e$ for some $k \in \{1, \ldots, r\}$. The induction assumption applied to this premiss gives us a derivation of $\Gamma' \vdash_m P_k[S/x] : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$. By applying back the (ND) rule we derive $\Gamma' \vdash_m R[S/x] : \hat{\tau} \triangleright c$, as required.

Next, suppose that $R = \lambda y.P$. We have $y \neq x$, and, as always during a substitution, we assume (by performing alpha-conversion) that y is not free in S. The derivation for R ends with the (λ) rule, whose premiss is $\Upsilon^x[y \mapsto C'] \vdash_m P : \hat{\tau}' \triangleright e$. While writing $\Upsilon^x[y \mapsto C']$ we mean that $\Upsilon^x(y) = \mathbf{0}$, and since y is not free in S, we have $\Delta_i(y) = \mathbf{0}$ for $i \in I$; thus we can write $\Gamma[y \mapsto C'] = \Upsilon^x[y \mapsto C'] \sqcup \bigsqcup_{i \in I} \Delta_i$. By applying the induction assumption to our premiss we obtain a derivation of $\Gamma'[y \mapsto C''] \vdash_m P[S/x] : \hat{\tau}' \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$ and some $C'' \sqsubseteq C'$. We then apply again the (λ) rule obtaining $\Gamma' \vdash_m R[S/x] : \hat{\tau} \triangleright c$, as needed.

Another possibility is that $R = a\langle P\rangle$, where $a \neq \mathsf{nd}$. Then the derivation for R ends with the (CON1) rule, whose premiss is $\Upsilon^x \vdash_m P : \hat{\tau}_1 \triangleright e_1$. We apply the induction assumption to this premiss, and we obtain a derivation of $\Gamma' \vdash_m P[S/x] : \hat{\tau}_1 \triangleright c_1$ for $c_1 = e_1 + \sum_{i \in I} d_i$ and for some $\Gamma' \sqsubseteq \Gamma$. To the obtained type judgment we apply the (CON1) rule, and we derive $\Gamma' \vdash_m R[S/x] : \hat{\tau} \triangleright c$ for $c = e + \sum_{i \in I} d_i$. We need to notice here that $c - e = c_1 - e_1$, and thus if $(F, e) \in \mathit{Comp}_m(M; (\{(1, a)\}, \mathbf{0}), (F_1, e_1))$ for some arguments M, F, F_1 (as ensured by the original use of the rule), then also $(F, c) \in \mathit{Comp}_m(M; (\{(1, a)\}, \mathbf{0}), (F_1, c_1))$ (as needed for the new use of the rule).

Finally, suppose that $R = P\,Q$. The derivation for R ends with the (@) rule, whose premisses are $\Upsilon^x_0 \vdash_m P : \hat{\tau}_0 \triangleright e_0$ and $\Upsilon^x_j \vdash_m Q : \hat{\tau}_j \triangleright e_j$ for $j \in J$, where we assume that $0 \notin J$. We have that $\Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}] = \Upsilon^x = \bigsqcup_{j \in \{0\} \cup J} \Upsilon^x_j$. Let us see that we can find sets $(I_j)_{j \in \{0\} \cup J}$ such that $I = \bigcup_{j \in \{0\} \cup J} I_j$ and $\{\!|\hat{\sigma}_i \mid i \in I_j|\!\} = \Upsilon^x_j(x)$ for all $j \in \{0\} \cup J$. Indeed, recall that triple containers behave like sets for balanced type triples, and like multisets for unbalanced type triples. Thus, if $\hat{\sigma}_i$ is balanced for some $i \in I$, we can simply add this i to I_j for all these $j \in \{0\} \cup J$ for which $\Upsilon^x_j(x)(\hat{\sigma}_i) > 0$. On the other hand, for an unbalanced type triple $\hat{\sigma}$, there exist exactly $\Upsilon^x(x)(\hat{\sigma})$ elements $i \in I$ for which $\hat{\sigma}_i = \hat{\sigma}$; simultaneously $\Upsilon^x(x)(\hat{\sigma}) = \sum_{j \in \{0\} \cup J} \Upsilon^x_j(x)(\hat{\sigma})$, so we can split these elements i into sets $(I_j)_{j \in \{0\} \cup J}$ so that exactly $\Upsilon^x_j(x)(\hat{\sigma})$ of them are taken to I_j (for $j \in \{0\} \cup J$).

Having these sets, for every $j \in \{0\} \cup J$ we can write $\Upsilon^x_j = \Upsilon_j[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I_j|\!\}]$. For these $i \in I$ for which the type triple $\hat{\sigma}_i$ is balanced, from Lemma 5.2 we obtain that all type triples in Δ_i are balanced, and

thus $\Delta_i \sqcup \Delta_i = \Delta_i$, and from Lemma 5.4 we obtain that $d_i = \mathbf{0}$. The latter lemma can be used because $ord(S) = ord(x) = m \leq m$. If we recall that every $i \in I$ with unbalanced $\hat{\sigma}_i$ belongs to exactly one among the sets I_j, and every $i \in I$ with balanced $\hat{\sigma}_i$ belongs to at least one among the sets I_j, we can observe that $\bigsqcup_{i\in I} \Delta_i = \bigsqcup_{j\in\{0\}\cup J} \bigsqcup_{i\in I_j} \Delta_i$ and $\sum_{i\in I} d_i = \sum_{j\in\{0\}\cup J} \sum_{i\in I_j} d_i$. In consequence, if we denote $\Gamma_j = \Upsilon_j \sqcup \bigsqcup_{i\in I_j} \Delta_i$ and $c_j = e_j + \sum_{i\in I_j} d_i$ for $j \in \{0\} \cup J$, and $c = e + \sum_{i\in I} d_i$, we have that $\Gamma = \bigsqcup_{j\in\{0\}\cup J} \Gamma_j$ and $c - e = \sum_{j\in\{0\}\cup J}(c_j - e_j)$. In particular $\Gamma_j(y) \leq \Gamma(y)$ for every $j \in \{0\} \cup J$ and every variable y, so Γ_j is a type environment (i.e., $\Gamma_j(x)$ contains every unbalanced type triple at most $|\Sigma|$ times).

We then apply the induction assumption to all premisses, and we obtain derivations of $\Gamma'_0 \vdash_m P[S/x] : \hat{\tau}_0 \triangleright c_0$ and of $\Gamma'_j \vdash_m Q[S/x] : \hat{\tau}_j \triangleright c_j$ for $j \in J$, where $\Gamma'_j \sqsubseteq \Gamma_j$ for $j \in \{0\} \cup J$. By applying the (@) rule again, we derive $\Gamma' \vdash_m R[S/x] : \hat{\tau} \triangleright c$ for $\Gamma' = \bigsqcup_{j\in\{0\}\cup J} \Gamma'_j \sqsubseteq \Gamma$ (the side conditions of the rule are satisfied, because we consider the same type triples as in the original derivation).

We also need to see that if some of the original derivations are wild, then the resulting derivation is wild as well. To this end, suppose that in the derivation of $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$ there is a wild use of the (@) rule. Then in the resulting derivation the (@) rule is used in a similar way, only the type environments and the considered lambda-terms are changed, but this is still a wild use of the (@) rule. Next, suppose that there is a wild use of the (@) rule in the derivation of $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for some $i \in I$. By Lemma 5.6 this can happen only when $\hat{\sigma}_i$ is unbalanced (recall that $ord(S) = m$, which allows us to use this lemma). This means that the derivation is inserted somewhere in the resulting derivation (we discard only derivations for balanced $\hat{\sigma}_i$), and the wild use of the (@) rule remains present. □

Proof of Lemma 7.1: Recall that we are given a derivation of $\Gamma \vdash_m P : \hat{\tau} \triangleright c$, and a beta-reduction $P \to_\beta Q$ that is of order m, and our goal is to derive $\Gamma' \vdash_m Q : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$.

Suppose first that $P = (\lambda x.R)\, S$ and $Q = R[S/x]$, where $ord(x) = m$. Then the given derivation ends with the (@) rule, whose premisses are $\Upsilon \vdash_m \lambda x.R : \hat{\tau}_\lambda \triangleright e$ and $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I$. Let us write $\hat{\tau} = (F, M, \tau)$, and $\hat{\tau}_\lambda = (F', M', C \to \tau)$, and $\hat{\sigma}_i = (F_i, M_i, \sigma_i)$ for $i \in I$. The type judgment for $\lambda x.R$ is in turn derived by the (λ) rule, whose premiss is $\Upsilon^x \vdash_m R : (F', M'', \tau) \triangleright e$, where $\Upsilon^x = \Upsilon[x \mapsto C']$ for some $C' \sqsubseteq C$, and $M' = M'' - \mathsf{Mk}(C)$. Because all F_i and M_i are m-bounded, and $ord(S) = ord(x) = m$, for all $i \in I$ we have that $F_i{\restriction}_{\leq ord(S)} = F_i$, and $M_i{\restriction}_{\leq ord(S)} = M_i$, and $F_i{\restriction}_{> ord(S)} = \emptyset$. Conditions of the (@) rule imply that

1. $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$,
2. $C = \{\!\{(F_i{\restriction}_{\leq ord(S)}, M_i{\restriction}_{\leq ord(S)}, \sigma_i) \mid i \in I\}\!\} = \{\!\{\hat{\sigma}_i \mid i \in I\}\!\}$,
3. $M = M' + \sum_{i\in I} M_i = M'' - \mathsf{Mk}(C) + \sum_{i\in I} M_i = M''$,
4. $(F, c) \in \mathit{Comp}_m(M; (F', e), ((F_i{\restriction}_{> ord(S)}, d_i))_{i\in I})$, which by Lemma 6.6 implies that $F \subseteq F'$ and $c = e + \sum_{i\in I} d_i$ (where $M(k) = M''(k) = 0$ for all $(k, a) \in F'$ because (F', M'', τ) is a type triple), and
5. $\{(k, a) \in F' \mid M(k) = 0\} \subseteq F$, so $F' = F$, and thus the type triple derived for R is actually $\hat{\tau}$.

Since $C' \sqsubseteq C$, we can find some $I' \subseteq I$ such that $C' = \{\!\{\hat{\sigma}_i \mid i \in I'\}\!\}$. Moreover, for every $i \in I \setminus I'$ the type triple $\hat{\sigma}_i$ is necessarily balanced, so $\mathsf{Mk}(\Delta_i) = \mathbf{0}$ by Lemma 5.2, and $d_i = \mathbf{0}$ by Lemma 5.4. In consequence, $\Upsilon \sqcup \bigsqcup_{i\in I'} \Delta_i \sqsubseteq \Gamma$ (in particular $\Upsilon \sqcup \bigsqcup_{i\in I'} \Delta_i$ is a type environment) and $c = e + \sum_{i\in I'} d_i$. We apply Lemma 7.4 to $\Upsilon^x \vdash_m R : \hat{\tau} \triangleright e$ and to $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I'$; we obtain a derivation of $\Gamma' \vdash_m Q : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Upsilon \sqcup \bigsqcup_{i\in I'} \Delta_i \sqsubseteq \Gamma$, as required.

We also need to see that if the original derivation was wild, then the new one is also wild. Notice that the final use of the (@) rule in the original derivation cannot be wild: for its wildness we would need an

element $(k,a) \in F_i{\restriction}_{>ord(Q)} = \emptyset$ for some $i \in I$. Moreover, the removed subderivations ending with $\Delta_i \vdash_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I \setminus I'$ cannot be wild by Lemma 5.6. Thus the wild use of the (@) rule is located in some of the subderivations passed to Lemma 7.4, and thus it is preserved.

It remains to consider the general situation: the redex involved in the beta-reduction $P \to_{\beta(m)} Q$ is located somewhere deeper in P. Then the proof is by an easy induction on the depth of this redex. In the induction step we apply the induction assumption to appropriate premisses of the final rule, and we observe that after applying it, the rule can still be used. For the (@) rule we need the trivial observation that if $\Gamma' \sqsubseteq \Gamma$ and $\Delta' \sqsubseteq \Delta$ then $\Gamma' \sqcup \Delta' \sqsubseteq \Gamma \sqcup \Delta$. We have to be slightly more careful only for the (λ) rule: if its premiss is $\Gamma[x \mapsto C'] \vdash_m R : (F, M, \tau) \triangleright c$ and its conclusion is $\Gamma \vdash_m \lambda x.R : (F, M - \mathsf{Mk}(C), C \to \tau) \triangleright c$, by the induction assumption we obtain a derivation of $\Gamma'[x \mapsto C''] \vdash_m S : (F, M, \tau) \triangleright c$ with $\Gamma' \sqsubseteq \Gamma$ and $C'' \sqsubseteq C' \sqsubseteq C$; we can then apply the (λ) rule and derive $\Gamma' \vdash_m S : (F, M - \mathsf{Mk}(C), C \to \tau) \triangleright c$. □

Remark. Recall that the (λ) rule allows to forget about some balanced type triples provided by an argument, that is, we can have $C' \sqsubseteq C$. We notice, however, that in derivations constructed in Section 6 we use the (λ) rule only for $C' = C$. This means that Theorem 3.2 holds also for a more restrictive type system in which the condition $C' \sqsubseteq C$ in the (λ) rule is replaced by $C' = C$. On the other hand, in Lemma 7.4 it is necessary to discard some type judgments for S (cf. the case of a variable), so the type environment Γ' in the resulting type judgment only satisfies $\Gamma' \sqsubseteq \Gamma$, not $\Gamma' = \Gamma$. In consequence, in surrounding (λ) rules $C' \sqsubseteq C$ starts to hold instead of $C' = C$. Thus, even if we start from a derivation in the more restrictive type system (i.e., with $C' = C$), in the soundness proof we pass through derivations in the original type system (i.e., with $C' \sqsubseteq C$).

7.2 Proof of Lemma 7.2

The proof is easy; we simply replace $\vdash_m$ by $\vdash_{m-1}$ in all derived type judgments, and we ignore flags of order $m+1$ and markers of order m. To obtain the inequality $c' \geq c$ we observe that when the complexity is at most m, information about flags of order m goes only from descendants to ancestors, and thus every flag of order $m+1$ is created because of a different flag of order m.

We now give more details. The first lemma describes the behavior of the $Comp_m$ predicate.

Lemma 7.5. *Suppose that $(F,c) \in Comp_m(M; ((F_i, c_i))_{i \in I})$, and $m \geq 1$, and that $M(k) = 0$ for all $(k,a) \in F$. Suppose also that for every $i \in I$ either*

- *$F_i \in \mathcal{F}_m$, and $F'_i = F_i{\restriction}_{\leq m-1}$, and $c'_i \colon \Sigma \to \mathbb{N}$ is such that $c'_i(a) \geq c_i(a) + |F_i \cap \{(m,a)\}|$ for all $a \in \Sigma$, or*
- *$(F_i, c_i) = (F'_i, c'_i) = (\{(1,a)\}, \mathbf{0})$ for some $a \in \Sigma$.*

Then $(F{\restriction}_{\leq m-1}, c') \in Comp_{m-1}(M{\restriction}_{\leq m-1}; ((F'_i, c'_i))_{i \in I})$ for some $c' \colon \Sigma \to \mathbb{N}$ such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$.

Proof: We consider the numbers $f_{k,a}$ and $f'_{k,a}$ appearing in the definition of $Comp_m(M; ((F_i, c_i))_{i \in I})$. We also consider analogous numbers defined by the predicate $Comp_{m-1}(M{\restriction}_{\leq m-1}; ((F'_i, c'_i))_{i \in I})$, and we call them $g_{k,a}$ and $g'_{k,a}$. Since the arguments are the same up to order $m-1$, for every $a \in \Sigma$ we have $g_{k,a} = f_{k,a}$ for $k \leq m-1$, and $g'_{k,a} = f'_{k,a}$ for $k \leq m$. In consequence, the requirements given by $Comp_{m-1}$ on the set F (i.e., that $g_{k,a} > 0$ for all $(k,a) \in F{\restriction}_{\leq m-1}$) follow directly from the requirements given by $Comp_m$ (saying that $f_{k,a} > 0$ for all $(k,a) \in F$). We take $c'(a) = g_{m,a} + \sum_{i \in I} c'_i(a)$ for all $a \in \Sigma$, as required by the definition of $Comp_{m-1}$.

It remains to prove that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. For the rest of the proof fix some $a \in \Sigma$. We have that

$$c'(a) = g_{m,a} + \sum_{i \in I} c'_i(a) = g'_{m,a} + \sum_{i \in I} |F'_i \cap \{(m,a)\}| + \sum_{i \in I} c'_i(a)\,.$$

For every $i \in I$ we have one of two cases: either

- $c'_i(a) \geq c_i(a) + |F_i \cap \{(m,a)\}|$, or
- $(F_i, c_i) = (F'_i, c'_i) = (\{(1,b)\}, \mathbf{0})$ for some $b \in \Sigma$.

In both cases we see that $|F'_i \cap \{(m,a)\}| + c'_i(a) \geq |F_i \cap \{(m,a)\}| + c_i(a)$. Recalling that $g'_{m,a} = f'_{m,a}$ we obtain that

$$c'(a) \geq f'_{m,a} + \sum_{i \in I} |F_i \cap \{(m,a)\}| + \sum_{i \in I} c_i(a) = f_{m,a} + \sum_{i \in I} c_i(a)\,.$$

Next, let us observe that $f_{m,a} \geq f'_{m+1,a} + |F \cap \{(m,a)\}|$. Indeed, if $M(m) > 0$, we have $f'_{m+1,a} = f_{m,a}$ and $(m,a) \notin F$. Conversely, if $M(m) = 0$, we have $f'_{m+1,a} = 0$, and if $f_{m,a} = 0$ then also $(m,a) \notin F$.

Moreover, because all F_i are m-bounded (recall that $m \geq 1$), we have that $f_{m+1,a} = f'_{m+1,a} + \sum_{i \in I} |F_i \cap \{(m+1,a)\}| = f'_{m+1,a}$. We thus obtain that

$$\begin{aligned} c'(a) &\geq f'_{m+1,a} + \sum_{i \in I} c_i(a) + |F \cap \{(m,a)\}| \\ &= f_{m+1,a} + \sum_{i \in I} c_i(a) + |F \cap \{(m,a)\}| = c(a) + |F \cap \{(m,a)\}|\,. \end{aligned}$$ □

The statement of Lemma 7.2 is not suitable for an inductive proof (it talks only about type judgments for closed lambda-terms of sort o). Thus, in order to prove this lemma, we now generalize it to arbitrary type judgments.

Lemma 7.6. *Let P be a lambda-term of complexity at most $m \geq 1$, whose all free variables are of order at most $m-1$. If we can derive $\Gamma \vdash_m P : (F, M, \tau) \triangleright c$, then we can also derive $\Gamma \vdash_{m-1} P : (F{\restriction}_{\leq m-1}, M{\restriction}_{\leq m-1}, \tau) \triangleright c'$ for some $c' \colon \Sigma \to \mathbb{N}$ such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. Moreover, if the original derivation was wild, then the resulting one is also wild.*

Proof: Denote $\hat{\tau} = (F, M, \tau)$ and $\hat{\sigma} = (F{\restriction}_{\leq m-1}, M{\restriction}_{\leq m-1}, \tau)$. The proof is by induction on the structure of some fixed derivation of $\Gamma \vdash_m P : \hat{\tau} \triangleright c$. We distinguish several cases depending on the shape of P.

Suppose first that P is a variable, $P = x$. Then the (VAR) rule used in the derivation implies that $\Gamma = \varepsilon[x \mapsto (F, M{\restriction}_{\leq ord(x)}, \tau)]$), and $c = \mathbf{0}$. By assumption of the lemma we have $ord(x) \leq m-1$, so $(M{\restriction}_{\leq m-1}){\restriction}_{\leq ord(x)} = M{\restriction}_{\leq ord(x)}$ and $F{\restriction}_{\leq m-1} = F$ (because F is $ord(x)$-bounded). In consequence, we can use the (VAR) rule to derive $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright \mathbf{0}$.

Next, suppose that $P = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$. Then the final (ND) rule has a premiss $\Gamma \vdash_m P_k : \hat{\tau} \triangleright c$ for some $k \in \{1, \ldots, r\}$. The induction assumption applied to this premiss gives us a derivation of $\Gamma \vdash_{m-1} P_k : \hat{\sigma} \triangleright c'$ with c' such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. We apply back the (ND) rule, obtaining $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright c'$.

Next, suppose that $P = \lambda x.Q$. Then the final (λ) rule has a premiss $\Gamma[x \mapsto C'] \vdash_m Q : (F, M', \tau') \triangleright c$, where $\tau = C \to \tau'$, and $M = M' - \mathsf{Mk}(C)$, and $C' \sqsubseteq C$. Using the induction assumption for our premiss

(which is allowed, because $ord(x) \leq ord(P) - 1 \leq m-1$) we obtain a derivation of $\Gamma[x \mapsto C'] \vdash_{m-1} Q : (F{\restriction}_{\leq m-1}, M'{\restriction}_{\leq m-1}, \tau') \triangleright c'$ with c' such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. Because C is $ord(x)$-bounded, and $ord(x) \leq m-1$, we have that $M'{\restriction}_{\leq m-1} - \mathsf{Mk}(C) = (M' - \mathsf{Mk}(C)){\restriction}_{\leq m-1} = M{\restriction}_{\leq m-1}$, so by applying back the (λ) rule we derive $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright c'$.

Next, suppose that $P = b\langle P_1\rangle$ with $b \neq \mathsf{nd}$. Then $\tau = o$, and the final (CON1) rule has a premiss $\Gamma_1 \vdash_m P_1 : (F_1, M, o) \triangleright c_1$. By the induction assumption, we can derive $\Gamma \vdash_{m-1} P_1 : (F_1{\restriction}_{\leq m-1}, M{\restriction}_{\leq m-1}, o) \triangleright c_1'$ for some c_1' such that $c_1'(a) \geq c_1(a) + |F_1 \cap \{(m,a)\}|$ for all $a \in \Sigma$. A side condition says that $(F, c) \in Comp_m(M; (\{(1,b)\}, \mathbf{0}), (F_1, c_1))$, and we need to see that $(F{\restriction}_{\leq m-1}, c') \in Comp_{m-1}(M{\restriction}_{\leq m-1}; (\{(1,b)\}, \mathbf{0}), (F_1{\restriction}_{\leq m-1}, c_1'))$ for some c' such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$; this follows directly from Lemma 7.5. Thus, we can apply back the (CON1) rule, and derive $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright c'$.

Suppose now that $P = b\langle\rangle$ with $b \neq \mathsf{nd}$. Then $\tau = o$, and $(F, c) \in Comp_m(M; (\{(1,b)\}, \mathbf{0}))$. From Lemma 7.5 it follows that $(F{\restriction}_{\leq m-1}, c') \in Comp_{m-1}(M{\restriction}_{\leq m-1}; (\{(1,b)\}, \mathbf{0}))$ for some c' such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. We apply the (CON0) rule, and we derive $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright c'$.

Finally, suppose that $P = Q\,R$. Then the final (@) rule has premisses $\Gamma' \vdash_m Q : (F', M', C \to \tau) \triangleright e$ and $\Gamma_i \vdash_m R : (F_i, M_i, \tau_i) \triangleright d_i$ for $i \in I$, where we have that $C = \{\!\{(F_i{\restriction}_{\leq ord(R)}, M_i{\restriction}_{\leq ord(R)}, \tau_i) \mid i \in I\}\!\}$. The induction assumption applied to all premisses gives us a derivation of $\Gamma' \vdash_{m-1} Q : (F'{\restriction}_{\leq m-1}, M'{\restriction}_{\leq m-1}, C \to \tau) \triangleright e'$ with e' such that $e'(a) \geq e(a) + |F' \cap \{(m,a)\}|$ for all $a \in \Sigma$, and, for all $i \in I$, a derivation of $\Gamma_i \vdash_{m-1} R : (F_i{\restriction}_{\leq m-1}, M_i{\restriction}_{\leq m-1}, \tau_i) \triangleright d_i'$ with d_i' such that $d_i'(a) \geq d_i(a) + |F_i \cap \{(m,a)\}|$ for all $a \in \Sigma$. The side condition $M = M' + \sum_{i\in I} M_i$ implies $M{\restriction}_{\leq m-1} = M'{\restriction}_{\leq m-1} + \sum_{i\in I} M_i{\restriction}_{\leq m-1}$, and the side condition $\{(k,a) \in F' \mid M(k) = 0\} \subseteq F$ implies $\{(k,a) \in F'{\restriction}_{\leq m-1} \mid M{\restriction}_{\leq m-1}(k) = 0\} \subseteq F{\restriction}_{\leq m-1}$. Another side condition says that $(F, c) \in Comp_m(M; (F', e), (F_i{\restriction}_{>ord(R)}, d_i)_{i\in I})$. Because the complexity of P is at most m, we have $ord(R) \leq ord(Q) - 1 \leq m-1$. In consequence, $d_i'(a) \geq d_i(a) + |F_i \cap \{(m,a)\}| = d_i(a) + |F_i{\restriction}_{>ord(R)} \cap \{(m,a)\}|$ for all $i \in I$ and $a \in \Sigma$, thus by Lemma 7.5 we obtain $(F{\restriction}_{\leq m-1}, c') \in Comp_{m-1}(M{\restriction}_{\leq m-1}; (F'{\restriction}_{\leq m-1}, e'), ((F_i{\restriction}_{>ord(R)}){\restriction}_{\leq m-1}, d_i')_{i\in I})$ for some c' such that $c'(a) \geq c(a) + |F \cap \{(m,a)\}|$ for all $a \in \Sigma$. Clearly, $(F_i{\restriction}_{>ord(R)}){\restriction}_{\leq m-1} = (F_i{\restriction}_{\leq m-1}){\restriction}_{>ord(R)}$. Moreover, $C = \{\!\{(F_i{\restriction}_{\leq m-1}){\restriction}_{\leq ord(R)}, (M_i{\restriction}_{\leq m-1}){\restriction}_{\leq ord(R)}, \tau_i) \mid i \in I\}\!\}$, again because $ord(R) \leq m-1$. Having all this, we can apply back the (@) rule, and derive $\Gamma \vdash_{m-1} P : \hat{\sigma} \triangleright c'$.

Still staying in the case of $P = Q\,R$, suppose now that the original derivation ends with a wild use of the (@) rule. This means that for some $i \in I$ and for some $(k,a) \in F_i{\restriction}_{>ord(R)}$ we have $M_i = \mathbf{0}$ and $M(l) > 0$ for all $l \in \{k, k+1, \ldots, m\}$. If $k = m$, the type judgment derived by the induction assumption, $\Gamma_i \vdash_{m-1} R : (F_i{\restriction}_{\leq m-1}, M_i{\restriction}_{\leq m-1}, \tau_i) \triangleright d_i'$, satisfies $M_i{\restriction}_{\leq m-1} = \mathbf{0}$ and $d_i'(a) \geq d_i(a) + |F_i \cap \{(m,a)\}| \geq 1$, which is impossible by Lemma 5.4 (we use here the fact that $ord(R) \leq m-1$). Thus $k < m$, so we have as well $(k,a) \in (F_i{\restriction}_{\leq m-1}){\restriction}_{>ord(R)}$. It is also true that $M_i{\restriction}_{\leq m-1} = \mathbf{0}$ and $M{\restriction}_{\leq m-1}(l) > 0$ for all $l \in \{k, k+1, \ldots, m-1\}$. Thus the (@) rule is used wildly also in the resulting derivation.

We notice that the proof does not remove any fragment of the original derivation. Thus, by the above, if the original derivations contains some wild use of the (@) rule, the resulting derivation also contains a wild use of the (@) rule, in the same place. □

Lemma 7.2 is obtained by specializing Lemma 7.6 to the situation when P is closed, and $(F, M, \tau) = \hat{\rho}_m^{all}$. Notice that then $(F{\restriction}_{\leq m-1}, M{\restriction}_{\leq m-1}, \tau) = \hat{\rho}_{m-1}^{all}$.

7.3 Proof of Lemma 7.3

In this lemma, we are given a derivation of $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$, where P is of complexity 0. The proof is by induction on the structure of some fixed derivation of $\varepsilon \vdash_0 P : \hat{\rho}_0^{all} \triangleright c$. Let us analyze the shape of P. Because the type environment is empty, and because P has complexity 0, P cannot be a variable, nor a lambda-binder, nor an application. Thus P starts with a node constructor. We have three cases.

The first case is that $P = \mathsf{nd}\langle P_1, \ldots, P_r\rangle$. Then the final (ND) rule has one premiss $\varepsilon \vdash_0 P_i : \hat{\rho}_0^{all} \triangleright c$ for some $i \in \{1, \ldots, r\}$. The induction assumption gives us a word T such that $P_i \to_{\mathsf{nd}}^* T$ and for every $a \in \Sigma$, the number of occurrences of a in T is $c(a)$. Since $P \to_{\mathsf{nd}} P_i$, this gives the thesis.

Suppose now that $P = b\langle P_1\rangle$. Observe that $\hat{\rho}_0^{all} = (\emptyset, \{\!\{0\}\!\}, o)$ is the only type triple in $\mathcal{TT}_0^o$. Indeed, if $(F, M, \tau) \in \mathcal{TT}_0^o$, then $\tau = o$, and F and M are 0-bounded. For F this implies that $F = \emptyset$, and for M this implies that $M(k) = 0$ for all $k \geq 1$. Additionally, the definition of a type triple requires that $M(0) = 1$. It follows that the premiss of the final (CON1) rule is of the form $\varepsilon \vdash_0 P_1 : \hat{\rho}_0^{all} \triangleright c_1$. By the induction assumption we obtain a tree T_1 such that $P_1 \to_{\mathsf{nd}}^* T_1$ and for every $a \in \Sigma$, the number of occurrences of a in T_1 is $c_1(a)$. We take $T = b\langle T_1\rangle$; then $P \to_{\mathsf{nd}}^* T$. As in the proof of Lemma 6.2, we observe that $(\emptyset, c) \in \mathit{Comp}_0(\mathsf{Mk}(\hat{\rho}_0^{all}); (\{(1, b)\}, \mathbf{0}), (\emptyset, c_1))$ holds exactly when $c(a) = c_1(a)$ for $a \in \Sigma \setminus \{b\}$, and $c(b) = 1 + c_1(b)$. It follows that for every $a \in \Sigma$, the number of occurrences of a in T is $c(a)$.

Finally, suppose that $P = b\langle\rangle$. Then $(\emptyset, c) \in \mathit{Comp}_0(\mathsf{Mk}(\hat{\rho}_0^{all}); (\{(1, b)\}, \mathbf{0}))$ implies that $c(b) = 1$ and $c(a) = 0$ for all $a \in \Sigma \setminus \{b\}$. Taking $T = b\langle\rangle$, we see that $P \to_{\mathsf{nd}}^* T$ and that for every $a \in \Sigma$, the number of occurrences of a in T is $c(a)$.

8 Between tree-recognizing and word-recognizing schemes

Recall that the type system presented in previous sections works only for word-recognizing schemes. It is not difficult to improve the type system and make it working also for tree-recognizing schemes, at the cost of increasing exponentially the number of possible types of each sort (see, e.g., Parys (2017b), where a similar type system is presented in a version that works for all schemes). In other words, if we want to solve SUP for word-recognizing schemes in the optimal complexity, it is necessary to have a type system that works only for word-recognizing schemes. Instead of presenting another variant of the type system, working for all schemes, we rather give a translation from tree-recognizing schemes to word-recognizing schemes. This translation is presented in the current section; it increases the order of the considered scheme by one, and increases its size only linearly. A translation similar to the one presented here is sketched in Asada and Kobayashi (2016).

We remark that Asada and Kobayashi (2016) give also a translation in the opposite direction: from word-recognizing schemes to tree-recognizing schemes of order lower by one. This translation, however, severely increases the size of a transformed scheme, and thus it cannot be used to solve SUP for word-recognizing schemes (by first reducing to a tree-recognizing scheme, and then using a type system for tree-recognizing schemes) in the optimal complexity.

The intuition behind the translation is that instead of generating a tree, we generate a word containing all nodes of the tree (ordered from left to right, with a node before its descendants).

Thus, for a finite tree T and a word W we define $\mathit{word}(T, W)$ by induction on the size of T:

$$\mathit{word}(a\langle T_1, \ldots, T_r\rangle, W) = a\langle \mathit{word}(T_1, \mathit{word}(T_2, \mathit{word}(\ldots, \mathit{word}(T_r, W)\ldots)))\rangle\,.$$

Moreover, when $\mathcal{L}_1$ is a set of finite trees, and $\mathcal{L}_2$ is a set of words, let $\mathit{word}(\mathcal{L}_1, \mathcal{L}_2)$ be the set containing $\mathit{word}(T, W)$ for all $T \in \mathcal{L}_1$ and $W \in \mathcal{L}_2$.

Next, we say how to transform sorts, lambda-terms, and schemes. For every sort α we define a sort $\alpha^\flat$, by induction on its size:

$$o^\flat = o \to o \qquad \text{and} \qquad (\beta \to \gamma)^\flat = \beta^\flat \to \gamma^\flat\,.$$

For every variable x of sort α (including all nonterminals), let $x^\flat$ be a corresponding variable of sort $\alpha^\flat$. For a lambda-term R we define $R^\flat$ by coinduction on its structure:

$$x^\flat = x^\flat\,, \qquad (P\,Q)^\flat = P^\flat\,Q^\flat\,, \qquad (\lambda x.P)^\flat = \lambda x^\flat.P^\flat\,,$$
$$(\mathsf{nd}\langle P_1,\dots,P_r\rangle)^\flat = \lambda \mathsf{z}.\mathsf{nd}\langle P_1^\flat\,\mathsf{z},\dots,P_r^\flat\,\mathsf{z}\rangle\,, \qquad \text{and}$$
$$(a\langle P_1,\dots,P_r\rangle)^\flat = \lambda \mathsf{z}.a\langle P_1^\flat\,(P_2^\flat\,(\dots\,(P_r^\flat\,\mathsf{z})\dots))\rangle \qquad \text{for } a \neq \mathsf{nd}\,,$$

where z is a fresh variable of sort o. For a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$ we define a scheme $\mathcal{G}^\flat$ to have nonterminals $N^\flat$ for all $N \in \mathcal{N}$ and a fresh starting nonterminal N_0' of sort o, and to have rules $\mathcal{R}^\flat(N^\flat) = (\mathcal{R}(N))^\flat$ and $\mathcal{R}^\flat(N_0') = N_0\,(\mathsf{e}\langle\rangle)$ (for a fixed letter e).

Example 8.1. Let $\mathcal{G}_\triangle = (\mathcal{N}_\triangle, \mathcal{R}_\triangle, \mathsf{M})$ be a tree-recognizing scheme (of order 1), where $\mathcal{N}_\triangle = \{\mathsf{M}, \mathsf{N}\}$ with M of sort o and N of sort $o \to o$, and where

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\mathsf{a}\langle\rangle)\,, \qquad \mathcal{R}(\mathsf{N}) = \lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{x}, \mathsf{N}\,(\mathsf{a}\langle \mathsf{x}, \mathsf{x}\rangle)\rangle\,.$$

We have $\mathcal{G}_\triangle^\flat = (\mathcal{N}_\triangle^\flat, \mathcal{R}_\triangle^\flat, \mathsf{M}')$, where $\mathcal{N}_\triangle^\flat = \{\mathsf{M}^\flat, \mathsf{N}^\flat, \mathsf{M}'\}$ with $\mathsf{M}^\flat$ of sort $o^\flat = o \to o$, and $\mathsf{N}^\flat$ of sort $(o \to o)^\flat = (o \to o) \to o \to o$, and M' of sort o, and where, for $\mathsf{x}^\flat$ of sort $o^\flat = o \to o$ and for z of sort o,

$$\mathcal{R}^\flat(\mathsf{M}^\flat) = (\mathcal{R}(\mathsf{M}))^\flat = \mathsf{N}^\flat\,(\lambda \mathsf{z}.\mathsf{a}\langle\mathsf{z}\rangle)\,, \qquad \mathcal{R}^\flat(\mathsf{M}') = \mathsf{M}^\flat\,(\mathsf{e}\langle\rangle)\,,$$
$$\mathcal{R}^\flat(\mathsf{N}^\flat) = (\mathcal{R}(\mathsf{N}))^\flat = \lambda \mathsf{x}^\flat.\lambda \mathsf{z}.\mathsf{nd}\langle \mathsf{x}^\flat\,\mathsf{z}, \mathsf{N}^\flat\,(\lambda \mathsf{z}.\mathsf{a}\langle \mathsf{x}^\flat\,(\mathsf{x}^\flat\,\mathsf{z})\rangle)\,\mathsf{z}\rangle\,.$$

Observe that $\mathcal{L}(\mathcal{G}_\triangle)$ contains balanced binary trees of every height $i \geq 1$ (the tree of height i has $2^i - 1$ a-labeled nodes). On the other hand, $\mathcal{L}(\mathcal{G}_\triangle^\flat)$ contains words consisting of $2^i - 1$ a-labeled nodes, followed by an e-labeled node, for every $i \geq 1$. We thus have $word(\mathcal{L}(\mathcal{G}_\triangle), e\langle\rangle) = \mathcal{L}(\mathcal{G}_\triangle^\flat)$ (for example, $word(\mathsf{a}\langle\mathsf{a}\langle\rangle, \mathsf{a}\langle\rangle\rangle, \mathsf{e}\langle\rangle) = \mathsf{a}\langle\mathsf{a}\langle\mathsf{a}\langle\mathsf{e}\langle\rangle\rangle\rangle\rangle$). □

We are now ready to state a lemma constituting the expected properties of the translation.

Lemma 8.1.

1. *If P is a closed lambda-term of sort o, then*
   (a) *$\mathcal{L}(BT(P^\flat\,(\mathsf{e}\langle\rangle))) = word(\mathcal{L}(BT(P)), \mathsf{e}\langle\rangle)$, and*
   (b) *for every set of letters A we have $SUP_A(\mathcal{L}(BT(P^\flat\,(\mathsf{e}\langle\rangle))))$ if and only if $SUP_A(\mathcal{L}(BT(P)))$;*
2. *if $\mathcal{G}$ is a scheme of order m, then*
   (a) *$\mathcal{G}^\flat$ is a word-recognizing scheme of order $m+1$,*
   (b) *$|\mathcal{G}^\flat| \leq 6|\mathcal{G}| + 4$ (and thus $\mathcal{G}^\flat$ can be constructed in logarithmic space),*
   (c) *for every lambda-term R it is the case that $(\Lambda_G(R))^\flat = \Lambda_{\mathcal{G}^\flat}(R^\flat)$,*
   (d) *$\mathcal{L}(\mathcal{G}^\flat) = word(\mathcal{L}(\mathcal{G}), \mathsf{e}\langle\rangle)$, and*
   (e) *for every set of letters A, we have $SUP_A(\mathcal{L}(\mathcal{G}))$ if and only if $SUP_A(\mathcal{L}(\mathcal{G}^\flat))$.*

In order to establish Lemma 8.1, we first say that the translation commutes with substitutions and beta-reductions.

Lemma 8.2. *For all lambda-terms R, S and for every variable x of the same sort as S it holds that $(R[S/x])^\flat = R^\flat[S^\flat/x^\flat]$.*

Proof: A trivial coinduction on the structure of R. □

Lemma 8.3. *If $P \xrightarrow{h}_\beta Q$ then $P^\flat \xrightarrow{h}_\beta Q^\flat$.*

Proof: By assumption $P = (\lambda x.R)\, S\, S_1\, \ldots\, S_s$ and $Q = R[S/x]\, S_1\, \ldots\, S_s$. Observe that

$$P^\flat = (\lambda x^\flat.R^\flat)\, S^\flat\, S_1^\flat\, \ldots\, S_s^\flat \xrightarrow{h}_\beta R^\flat[S^\flat/x^\flat]\, S_1^\flat\, \ldots\, S_s^\flat = (R[S/x])^\flat\, S_1^\flat\, \ldots\, S_s^\flat = Q^\flat\,,$$

where the equality $R^\flat[S^\flat/x^\flat] = (R[S/x])^\flat$ holds by Lemma 8.2. □

We now give two lemmata, implying directly the two inclusions corresponding to the equality from Lemma 8.1(1a).

Lemma 8.4. *If P, Q are closed lambda-terms of sort o, where Q is word-recognizing, and if T is a finite Σ-labeled tree such that $BT(P) \to^n_{\mathsf{nd}} T$, then $word(T, \mathcal{L}(BT(Q))) \subseteq \mathcal{L}(BT(P^\flat\, Q))$.*

Proof: The proof is by induction on $n + |T|$. Because $BT(P) \to^n_{\mathsf{nd}} T$, we can be sure that $BT(P) \neq \mathsf{nd}\langle\rangle$, and thus $P \xrightarrow{h}{}^*_\beta a\langle P_1, \ldots, P_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$ and some lambda-terms $P_1, \ldots, P_r$ such that $BT(P) = a\langle BT(P_1), \ldots, BT(P_r)\rangle$. By Lemma 8.3 (applied to every head beta-reduction in the sequence witnessing that $P \xrightarrow{h}{}^*_\beta a\langle P_1, \ldots, P_r\rangle$) we have that $P^\flat \xrightarrow{h}{}^*_\beta (a\langle P_1, \ldots, P_r\rangle)^\flat$, hence also $P^\flat\, Q \xrightarrow{h}{}^*_\beta (a\langle P_1, \ldots, P_r\rangle)^\flat\, Q$. We have two cases.

Suppose first that $a \neq \mathsf{nd}$. In this case $BT(P) \to^n_{\mathsf{nd}} T$ implies that $T = a\langle T_1, \ldots, T_r\rangle$, where $BT(P_i) \to^{n_i}_{\mathsf{nd}} T_i$ with $n_i \leq n$ and $|T_i| < |T|$ (i.e., $n_i + |T_i| < n + |T|$) for all $i \in \{1, \ldots, r\}$. Denote $L_r = \mathcal{L}(BT(Q))$ and $Q_r = Q$, and then consecutively for $i = r, r-1, \ldots, 1$ denote $L_{i-1} = word(T_i, L_i)$ and $Q_{i-1} = P_i^\flat\, Q_i$. By a (reversed) internal induction we prove that $L_i \subseteq \mathcal{L}(BT(Q_i))$ for all $i \in \{0, \ldots, r\}$. For $i = r$ this amounts to showing that $\mathcal{L}(BT(Q)) \subseteq \mathcal{L}(BT(Q))$, which is trivial. Assume now that the thesis holds for some $i \geq 1$, that is, that $L_i \subseteq \mathcal{L}(BT(Q_i))$. This implies that $L_{i-1} = word(T_i, L_i) \subseteq word(T_i, \mathcal{L}(BT(Q_i))$. Due to $BT(P_i) \to^{n_i}_{\mathsf{nd}} T_i$, the induction assumption (of the external induction, used with Q_i as Q) implies that $word(T_i, \mathcal{L}(BT(Q_i))) \subseteq \mathcal{L}(BT(P_i^\flat\, Q_i)) = \mathcal{L}(BT(Q_{i-1}))$, which altogether gives us the required thesis $L_{i-1} \subseteq \mathcal{L}(BT(Q_{i-1}))$. In particular, we have $L_0 \subseteq \mathcal{L}(BT(Q_0))$. This means that for every $W \in \mathcal{L}(BT(Q))$,

$$word(T_1, word(T_2, word(\ldots, word(T_r, W)\ldots))) \in \mathcal{L}(BT(Q_0))\,.$$

By appending a node-constructor (for the letter a) in the front, we obtain that

$$a\langle word(T_1, word(T_2, word(\ldots, word(T_r, W)\ldots)))\rangle \in \mathcal{L}(a\langle BT(Q_0)\rangle)\,,$$

hence also $word(T, W) \in \mathcal{L}(a\langle BT(Q_0)\rangle)$, because by definition

$$word(T, W) = word(a\langle T_1, \ldots, T_r\rangle, W) = a\langle word(T_1, word(T_2, word(\ldots, word(T_r, W)\ldots)))\rangle\,.$$

The above holds for every $W \in \mathcal{L}(BT(Q))$, so it implies that $word(T, \mathcal{L}(BT(Q))) \in \mathcal{L}(a\langle BT(Q_0)\rangle)$. Recall now that

$$P^\flat\, Q \xrightarrow{h}{}^*_\beta (a\langle P_1, \ldots, P_r\rangle)^\flat\, Q = (\lambda \mathsf{z}.a\langle P_1^\flat\, (P_2^\flat\, (\ldots\, (P_r^\flat\, \mathsf{z})\ldots))\rangle)\, Q \xrightarrow{h}_\beta a\langle P_1^\flat\, (P_2^\flat\, (\ldots\, (P_r^\flat\, Q)\ldots))\rangle$$
$$= a\langle Q_0\rangle\,,$$

meaning that $BT(P^\flat\, Q) = a\langle BT(Q_0)\rangle$; we thus have $word(T, \mathcal{L}(BT(Q))) \subseteq \mathcal{L}(BT(P^\flat\, Q))$, as we wanted to prove.

It remains to consider the situation when $a = \mathsf{nd}$. In this case $BT(P) \to_{\mathsf{nd}} BT(P_i) \to_{\mathsf{nd}}^{n-1} T$ for some $i \in \{1, \ldots, r\}$. The induction assumption implies that $word(T, \mathcal{L}(BT(Q))) \subseteq \mathcal{L}(BT(P_i^\flat\, Q))$. Additionally,

$$P^\flat\, Q \xrightarrow{h}{}_\beta^* (\mathsf{nd}\langle P_1, \ldots, P_r\rangle)^\flat\, Q = (\lambda \mathsf{z}.\mathsf{nd}\langle P_1^\flat\, \mathsf{z}, \ldots, P_r^\flat\, \mathsf{z}\rangle)\, Q \xrightarrow{h}{}_\beta \mathsf{nd}\langle P_1^\flat\, Q, \ldots, P_r^\flat\, Q\rangle\,,$$

meaning that $BT(P^\flat\, Q) = \mathsf{nd}\langle BT(P_1^\flat\, Q), \ldots, BT(P_r^\flat\, Q)\rangle$. In particular $BT(P^\flat\, Q) \to_{\mathsf{nd}} BT(P_i^\flat\, Q)$, which implies that $\mathcal{L}(BT(P_i^\flat\, Q)) \subseteq \mathcal{L}(BT(P^\flat\, Q))$. We thus again have $word(T, \mathcal{L}(BT(Q))) \subseteq \mathcal{L}(BT(P^\flat\, Q))$, as needed. □

Lemma 8.5. *If P, Q are closed lambda-terms of sort o, where Q is word-recognizing, and if W is a finite Σ-labeled word such that $BT(P^\flat\, Q) \to_{\mathsf{nd}}^n W$, then $W \in word(\mathcal{L}(BT(P)), W')$ for some finite Σ-labeled word W' and some number n' such that $BT(Q) \to_{\mathsf{nd}}^{n'} W'$ and $n' + |W'| < n + |W|$.*

Proof: The proof is by induction on $n + |W|$. Suppose first that there exists an infinite sequence of head beta-reductions $P = P_0 \xrightarrow{h}{}_\beta P_1 \xrightarrow{h}{}_\beta P_2 \xrightarrow{h}{}_\beta \ldots$. In such a case, by Lemma 8.3 we have as well that $P_0^\flat \xrightarrow{h}{}_\beta P_1^\flat \xrightarrow{h}{}_\beta P_2^\flat \xrightarrow{h}{}_\beta \ldots$, thus also $P_0^\flat\, Q \xrightarrow{h}{}_\beta P_1^\flat\, Q \xrightarrow{h}{}_\beta P_2^\flat\, Q \xrightarrow{h}{}_\beta \ldots$. In consequence, there is no sequence of head beta-reductions from $P^\flat\, Q$ to a lambda-term starting with a node constructor, which implies that $BT(P^\flat\, Q) = \mathsf{nd}\langle\rangle$. This contradicts the assumption $BT(P^\flat\, Q) \to_{\mathsf{nd}}^n W$, finishing the proof in this case.

Next, suppose that $P \xrightarrow{h}{}_\beta^* R$ for a lambda-term R such that $R \xrightarrow{h}{}_\beta S$ does not hold for any S, but S does not start with a node constructor. Because P (hence also R) is closed and of order o, this can only happen when R is an infinite application, $R = \ldots\, R_3\, R_2\, R_1$. Using Lemma 8.3 and appending Q, we obtain that $P^\flat\, Q \xrightarrow{h}{}_\beta^* R^\flat\, Q = \ldots\, R_3^\flat\, R_2^\flat\, R_1^\flat\, Q$. In this case we again have that there is no sequence of head beta-reductions from $P^\flat\, Q$ to a lambda-term starting with a node constructor, which implies that $BT(P^\flat\, Q) = \mathsf{nd}\langle\rangle$, contrary to the assumption $BT(P^\flat\, Q) \to_{\mathsf{nd}}^n W$.

The remaining (and the only interesting) case is when $P \xrightarrow{h}{}_\beta^* a\langle P_1, \ldots, P_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$ and some lambda-terms $P_1, \ldots, P_r$ such that $BT(P) = a\langle BT(P_1), \ldots, BT(P_r)\rangle$. As previously, using Lemma 8.3 and appending Q, we obtain that $P^\flat\, Q \xrightarrow{h}{}_\beta^* (a\langle P_1, \ldots, P_r\rangle)^\flat\, Q$. We have two cases.

Suppose first that $a \neq \mathsf{nd}$. Similarly to the previous proof, taking $Q_r = Q$ and $Q_{i-1} = P_i^\flat\, Q_i$ consecutively for $i = r, r-1, \ldots, 1$, we have that $BT(P^\flat\, Q) = a\langle BT(Q_0)\rangle$. Denote $n_0 = n$. The assumption $BT(P^\flat\, Q) \to_{\mathsf{nd}}^n W$ implies that $W = a\langle W_0\rangle$ for a word W_0 such that $BT(Q_0) \to_{\mathsf{nd}}^{n_0} W_0$; notice that $n_0 + |W_0| = n + |W| - 1 < n + |W|$. Next, we consider consecutive $i \in \{1, \ldots, r\}$, and assuming that we have W_{i-1} and n_{i-1} satisfying $BT(Q_{i-1}) \to_{\mathsf{nd}}^{n_{i-1}} W_{i-1}$ and $n_{i-1} + |W_{i-1}| < n + |W|$, we construct W_i and n_I satisfying $BT(Q_i) \to_{\mathsf{nd}}^{n_i} W_i$ and $n_i + |W_i| < n + |W|$ and $W_{i-1} \in word(\mathcal{L}(BT(P_i)), W_i)$. To this end, we recall that $Q_{i-1} = P_i^\flat\, Q_i$, thus the induction assumption (which can be used for $P_i, Q_i, W_{i-1}, n_{i-1}$ as P, Q, W, n, respectively, because $n_{i-1} + |W_{i-1}| < n + |W|$) gives us W_i and n_i such that $W_{i-1} \in word(\mathcal{L}(BT(P_i)), W_i)$ and $BT(Q_i) \to_{\mathsf{nd}}^{n_i} W_i$, and $n_i + |W_i| < n_{i-1} + |W_{i-1}| < n + |W|$. Having all the words W_i and numbers n_i, as W' we take W_r, and as n' we take n_r. We already know that $BT(Q) = BT(Q_r) \to_{\mathsf{nd}}^{n'} W'$ and $n' + |W'| < n + |W|$. The conditions $W_{i-1} \in word(\mathcal{L}(BT(P_i)), W_i)$ imply that for every $i \in \{1, \ldots, r\}$ there exists a tree $T_i \in \mathcal{L}(BT(P_i))$

such that $W_{i-1} = word(T_i, W_i)$. Observe that

$$W = a\langle word(T_1, word(T_2, word(\dots, word(T_r, W')\dots)))\rangle = word(a\langle T_1, \dots, T_r\rangle, W')\,,$$

and that $a\langle T_1, \dots, T_r\rangle \in \mathcal{L}(BT(P))$ (because $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$). In consequence, $W \in word(\mathcal{L}(BT(P)), W')$, as needed.

The second case is that $a = \mathsf{nd}$. As in the previous proof, we obtain that $BT(P^\flat\, Q) = \mathsf{nd}\langle BT(P_1^\flat\, Q), \dots, BT(P_r^\flat\, Q)\rangle$, thus the assumption $BT(P^\flat\, Q) \to_{\mathsf{nd}}^n W$ implies that $BT(P_i^\flat\, Q) \to_{\mathsf{nd}}^{n-1} W$ for some $i \in \{1, \dots, r\}$. The induction assumption implies gives us a finite Σ-labeled word W' and a number n' such that $W \in word(\mathcal{L}(BT(P_i)), W')$, and $BT(Q) \to_{\mathsf{nd}}^{n'} W'$, and $n' + |W'| < n + |W|$. Because $\mathcal{L}(BT(P_i)) \subseteq \mathcal{L}(\mathsf{nd}\langle BT(P_1), \dots, BT(P_r)\rangle) = \mathcal{L}(BT(P))$, we also have that $W \in word(\mathcal{L}(BT(P)), W')$, as needed. □

Finally, we bound the size of lambda-terms after the translation.

Lemma 8.6. *If P is a finite lambda-term, then $|P^\flat| \leq 6|P| - 2$.*

Proof: We first prove that $|\alpha^\flat| = 2|\alpha| + 1$ for every sort α, by induction on the size of α. Indeed, $|o^\flat| = |o \to o| = 3 = 2|o| + 1$, and $|(\beta \to \gamma)^\flat| = |\beta^\flat| + 1 + |\gamma^\flat| = 2(|\beta| + 1 + |\gamma|) + 1 = 2|\beta \to \gamma| + 1$.

We now prove the statement of the lemma, by induction on the size of P. A simple calculation shows the thesis for each of the possible forms of P:

- if $P = x$, then $|x^\flat| = 1 \leq 4 = 6|P| - 2$;
- if $P = Q\,R$, then $|P^\flat| = |Q^\flat| + 1 + |R^\flat| \leq 6|Q| - 2 + 1 + 6|R| - 2 \leq 6(|Q| + 1 + |R|) - 2 = 6|P| - 2$;
- if $P = \lambda x.Q$ for a variable x of sort α, then

$$|P^\flat| = |\lambda x^\flat.Q^\flat| = |\alpha^\flat| + 1 + |Q^\flat| \leq 2|\alpha| + 2 + 6|Q| - 2 \leq 6(|\alpha| + 1 + |Q|) - 2 = 6|P| - 2\,;$$

- if $P = \mathsf{nd}\langle P_1, \dots, P_r\rangle$, then

$$\begin{aligned} |P^\flat| &= |\lambda \mathsf{z}.\mathsf{nd}\langle P_1^\flat\, \mathsf{z}, \dots, P_r^\flat\, \mathsf{z}\rangle| = |o| + 1 + 1 + \textstyle\sum_{i=1}^r (|P_i^\flat| + 1 + 1) \\ &\leq 3 + \textstyle\sum_{i=1}^r (6|P_i| - 2 + 2) \leq 6(1 + \sum_{i=1}^r |P_i|) - 2 = 6|P| - 2\,; \end{aligned}$$

- if $P = a\langle P_1, \dots, P_r\rangle$ for $a \neq \mathsf{nd}$, then

$$\begin{aligned} |P^\flat| &= |\lambda \mathsf{z}.a\langle P_1^\flat\, (P_2^\flat\, (\dots\, (P_r^\flat\, \mathsf{z})\dots))\rangle = |o| + 1 + 1 + \textstyle\sum_{i=1}^r (|P_i^\flat| + 1) + 1 \\ &\leq 4 + \textstyle\sum_{i=1}^r (6|P_i| - 2 + 1) \leq 6(1 + \sum_{i=1}^r |P_i|) - 2 = 6|P| - 2\,. \end{aligned}$$

□

We are now ready to prove Lemma 8.1.

Proof of Lemma 8.1: In order to obtain Point 1(a), consider a closed lambda-term P of sort o. Every element W of $\mathcal{L}(BT(P^\flat\, (\mathsf{e}\langle\rangle)))$ is a finite Σ-labeled word, and satisfies $BT(P^\flat\, (\mathsf{e}\langle\rangle)) \to_{\mathsf{nd}}^n W$ for some number $n \in \mathbb{N}$. By Lemma 8.5, there is a word W' such that $BT(\mathsf{e}\langle\rangle) \to_{\mathsf{nd}}^* W'$ and $W \in word(\mathcal{L}(BT(P)), W')$. The only possibility is that $W' = \mathsf{e}\langle\rangle$; thus $W \in word(\mathcal{L}(BT(P)), \mathsf{e}\langle\rangle)$, which establishes the inclusion $\mathcal{L}(BT(P^\flat\, (\mathsf{e}\langle\rangle))) \subseteq word(\mathcal{L}(BT(P)), \mathsf{e}\langle\rangle)$. For the opposite inclusion, take some $W \in word(\mathcal{L}(BT(P)), \mathsf{e}\langle\rangle)$. By definition there is $T \in \mathcal{L}(BT(P))$ such that $W = word(T, \mathsf{e}\langle\rangle) \in word(T, \mathcal{L}(BT(\mathsf{e}\langle\rangle)))$; moreover, T is a finite Σ-labeled tree such that $BT(P) \to_{\mathsf{nd}}^n T$

for some $n \in \mathbb{N}$. Lemma 8.4 implies that $W \in \mathcal{L}(BT(P^\flat\,(e\langle\rangle)))$, which finishes the proof of the equality $\mathcal{L}(BT(P^\flat\,(\mathsf{e}\langle\rangle))) = word(\mathcal{L}(BT(P)), \mathsf{e}\langle\rangle)$.

Next, observe that $word(T, \mathsf{e})$ (for any finite Σ-labeled tree T) contains exactly the same letters as T, in the same quantity, except that it has one more occurrence of the letter e. In consequence, for every language L of finite Σ-labeled trees, and for every set of letters A, we have $SUP_A(word(L, \mathsf{e}\langle\rangle))$ if and only if $SUP_A(L)$. Thanks to this property, Point 1(b) follows from Point 1(a), and in the same way Point 2(e) follows from Point 2(d).

For Point 2, consider a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$. It follows directly from the definition that $\mathcal{G}^\flat$ has order higher by one, and that it is word-recognizing (Point 2(a)). Point 2(b), bounding the size of the resulting scheme $\mathcal{G}^\flat = (\mathcal{N}^\flat, \mathcal{R}^\flat, N_0')$, is obtained by the following calculation, using Lemma 8.6:

$$\begin{aligned}|\mathcal{G}^\flat| &= |o| + |\mathcal{R}^\flat(N_0')| + \textstyle\sum_{N^\alpha \in \mathcal{N}}(|\alpha^\flat| + |\mathcal{R}^\flat(N^\flat)|) = 1 + |N_0\,(\mathsf{e}\langle\rangle)| + \sum_{N^\alpha \in \mathcal{N}}(|\alpha^\flat| + |(\mathcal{R}(N))^\flat|)\\ &\leq 4 + \textstyle\sum_{N^\alpha \in \mathcal{N}}(2|\alpha| + 1 + 6|\mathcal{R}(N)| - 2) \leq 4 + 6\sum_{N^\alpha \in \mathcal{N}}(|\alpha| + |\mathcal{R}(N)|) = 6|\mathcal{G}| + 4\,.\end{aligned}$$

Moreover, it should be clear that, by following the straightforward definition, $\mathcal{G}^\flat$ can be constructed in logarithmic space.

Point 2(c) follows by a trivial coinduction using the definition of $\mathcal{G}^\flat$.

Finally, the equality $\mathcal{L}(\mathcal{G}^\flat) = word(\mathcal{L}(\mathcal{G}), \mathsf{e}\langle\rangle)$ from Point 2(d) is deduced from Point 1(a) after observing that $\Lambda(\mathcal{G}^\flat) = (\Lambda_{\mathcal{G}}(N_0))^\flat\,(e\langle\rangle)$ (which implies that $\mathcal{L}(\mathcal{G}^\flat) = \mathcal{L}(BT((\Lambda_{\mathcal{G}}(N_0))^\flat\,(e\langle\rangle)))$), and that $word(\mathcal{L}(\mathcal{G}), \mathsf{e}\langle\rangle) = word(\mathcal{L}(BT(\Lambda_{\mathcal{G}}(N_0))), \mathsf{e}\langle\rangle)$. □

9 Complexity of SUP

Using our type system we now establish the complexity of SUP for schemes, which is as described by the following theorem.

Theorem 9.1. *Let $m \in \mathbb{N}$.*

1. *If $m \geq 1$, SUP for tree-recognizing order-m schemes is m-EXPTIME-complete. If $m = 0$, it is NP-complete, and it is in FPT when $|A|$ is viewed as a parameter.*
2. *If $m \geq 2$, SUP for word-recognizing order-m schemes is $(m-1)$-EXPTIME-complete. If $m \in \{0, 1\}$, it is NP-complete, and it is in FPT when $|A|$ is viewed as a parameter.*

In the rest of this section we prove the above theorem. In the first part we concentrate on the upper bounds; the lower bounds are shown in Section 9.4 by easy reductions.

Concerning the upper bounds, recall at the beginning that for every (tree-recognizing) scheme $\mathcal{G}$ we can construct, in logarithmic space, a word-recognizing scheme $\mathcal{G}^\flat$ such that solving SUP for $\mathcal{G}$ boils down to solving SUP for $\mathcal{G}^\flat$; the order of the scheme grows by one (cf. Lemma 8.1, Point 2). As a result, all the upper bounds for tree-recognizing schemes (i.e., m-EXPTIME, NP, and FPT, as given by Point 1 of Theorem 9.1) are immediate consequences of the corresponding upper bounds for word-recognizing schemes (as given by Point 2 of Theorem 9.1).

In the proof of the upper bounds we thus assume that the considered scheme is word-recognizing; moreover, for compatibility with previous sections, we assume that it is of order $m + 1$, rather than of order m. We are thus given a set A and a word-recognizing scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$ of order at most $m + 1$, and we want to decide whether $SUP_A(\mathcal{L}(\mathcal{G}))$ holds. This should be done in m-EXPTIME for $m \geq 1$, and

in NP for $m = 0$. For $m = 0$ we also prove that the problem is fixed-parameter tractable when $|A|$ is viewed as a parameter.

Due to Theorem 3.2, solving SUP boils down to checking whether for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m^{all} \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$ (here we use the trivial fact that the complexity of $\Lambda(\mathcal{G})$ is not greater than the order of $\mathcal{G}$).

Before starting the proof, let us give three definitions.

Definition 9.2. Two type judgments are *equivalent* if they differ only in values of the flag counter.

Definition 9.3. For a set $A \subseteq \Sigma$, a derivation is *A-pumpable* if for every letter $a \in A$, there are two equivalent type judgments lying on one branch of the derivation and such that their flag counters differ on the a coordinate.

Definition 9.4. We say that a type judgment $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ is *useful* (with respect to a scheme $\mathcal{G}$) if Q is a subterm of $\Lambda(\mathcal{G})$ and $\Gamma(x) \neq \mathbf{0}$ only for variables x that appear in $\Lambda(\mathcal{G})$.

Let $\mathcal{U}^{\mathcal{G}}$ be the set of useful type judgments $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ satisfying $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$,[(iv)] and let $\mathcal{U}^{\mathcal{G}}_{/\sim}$ be the set of equivalence classes of type judgments from $\mathcal{U}^{\mathcal{G}}$.

For every rule of the type system it is easy to see that if the conclusion is useful, then also premisses are useful. Moreover, Lemma 5.2 tells us that all type judgments that can be derived satisfy the inequality $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$. It follows that all derivations of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m^{all} \triangleright c$ contain only type judgments from $\mathcal{U}^{\mathcal{G}}$.

We now proceed as follows. First, in Section 9.1, we bound the number of equivalence classes in $\mathcal{U}^{\mathcal{G}}_{/\sim}$. Then, in Section 9.2, we observe that if a flag counter c is large enough for every letter $a \in A$, then a derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m^{all} \triangleright c$ needs to be A-pumpable. Moreover, the opposite implication also holds: if we have an A-pumpable type derivation, then we can repeat (as many times as we want) its fragments between all pairs of equivalent type judgments, increasing arbitrarily the flag counter for all $a \in A$ in the resulting type judgment. This is described in Lemma 9.9. Finally, in Section 9.3 we give an algorithm exploiting this property.

9.1 Number of equivalence classes

In the first part, we bound the size of $\mathcal{U}^{\mathcal{G}}_{/\sim}$.

Recall that by $\Lambda_{\mathcal{G}}(P)$ we denote the lambda-term obtained by recursively expanding all nonterminals in a lambda-term P (which could contain nonterminals). It is easy to see that every subterm of $\Lambda(\mathcal{G})$ equals $\Lambda_{\mathcal{G}}(P)$ for some subterm P of $\mathcal{R}(N)$ for some nonterminal $N \in \mathcal{N}$. In consequence, there are at most $|\mathcal{G}|$ subterms of $\Lambda(\mathcal{G})$.

Let $\mathcal{S}_{\mathcal{G}}$ be the set of sorts of all subterms of $\mathcal{R}(N)$ for all nonterminals N of $\mathcal{G}$, and of all subsorts of these sorts (where we say that a sort α is a subsort of a sort β either if $\alpha = \beta$ or if $\beta = \gamma \to \delta$ and α is a subsort of γ or δ). Notice that the sorts of all subterms of $\Lambda(\mathcal{G})$ also belong to $\mathcal{S}_{\mathcal{G}}$. Let us first bound the size of sorts in $\mathcal{S}_{\mathcal{G}}$.

Lemma 9.5. *It is the case that $|\alpha| \leq 2 \cdot |\mathcal{G}|$ for every scheme $\mathcal{G}$ and every sort $\alpha \in \mathcal{S}_{\mathcal{G}}$.*[(v)]

[(iv)] As one can see in the proof, the inequality $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$ is important only when $m = 0$.

[(v)] One may wonder why we prove that $|\alpha| \leq 2 \cdot |\mathcal{G}|$ instead of $|\alpha| \leq |\mathcal{G}|$, but it is not clear whether the stronger inequality is always true. Surely an analogous inequality for lambda-terms is false: for example, the lambda-term $\lambda x^{\alpha}.x^{\alpha}$ is of size $2 + |\alpha|$, while its sort $\alpha \to \alpha$ is of size $2 \cdot |\alpha| + 1$.

Proof: We prove that if P of sort α is a subterm of $\mathcal{R}(N)$ for some nonterminal $N \in \mathcal{N}$, then $|\alpha| \leq |P| + |\mathcal{G}|$ (which clearly implies that $|\alpha| \leq 2 \cdot |\mathcal{G}|$ for all $\alpha \in \mathcal{S}_{\mathcal{G}}$). This is an induction on the size of P. When P starts with a node constructor this is trivial, since $\alpha = o$. When $P = Q\,R$, this follows directly from the inequality $|\beta \to \alpha| \leq |Q| + |\mathcal{G}|$ obtained from the induction assumption, where $\beta \to \alpha$ is the sort of Q, since $|\alpha| < |\beta \to \alpha|$ and $|Q| < |P|$. When $P = \lambda x.Q$, where $\alpha = \beta \to \gamma$, we obtain the thesis by adding $1 + |\beta|$ to both sides of the induction assumption $|\gamma| \leq |Q|$. When $P = x$ is a variable bound by some $\lambda x.Q$ somewhere in $\mathcal{R}(N)$, we have $|\alpha| \leq |\lambda x.Q| \leq |\mathcal{R}(N)| \leq |P| + |\mathcal{G}|$. Finally, $P = M$ may be a nonterminal, in which case $|\alpha|$ is also included in $|\mathcal{G}|$. □

Next, for $n \in \mathbb{N}$, denote by $\eta_n^{\mathcal{G}}$ the maximum of $|\mathcal{TT}_k^\alpha|$ over all $k \in \{0, \ldots, n\}$ and over all sorts $\alpha \in \mathcal{S}_{\mathcal{G}}$ such that $ord(\alpha) \leq n$. The next lemma bounds $\eta_n^{\mathcal{G}}$.

Lemma 9.6. *Let $n \in \mathbb{N}$. The number $\eta_n^{\mathcal{G}}$ is at most $(n-1)$-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$ when $n \geq 2$, and polynomial in $|\mathcal{G}|$ and $|\Sigma|$ when $n \leq 1$.*

Proof: In order to bound $\eta_n^{\mathcal{G}}$ (for particular values of n), take a number $k \in \{0, \ldots, n\}$ and a sort $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o \in \mathcal{S}_{\mathcal{G}}$ with $ord(\alpha) \leq n$ for which $\eta_n^{\mathcal{G}} = |\mathcal{TT}_k^\alpha|$, that is, for which the maximum is achieved. A type triple in $\mathcal{TT}_k^\alpha$ contains a flag set F, a marker multiset M, and a type $C_1 \to \ldots \to C_s \to o$. We have three cases:

- The only sort satisfying $ord(\alpha) \leq 0$ is $\alpha = o$. We have already observed in the proof of Lemma 7.3 that $\mathcal{TT}_0^o$ contains only one element, namely $\hat{\rho}_0^{all}$; thus $\eta_0^{\mathcal{G}} = 1$.
- Suppose now that $n = 1$. Then $\alpha_i = o$ for every $i \in \{1, \ldots, s\}$, so there is only one type triple that can be contained in the triple container C_i, namely $\hat{\rho}_0^{all}$, the only element of $\mathcal{TT}_{ord(\alpha_i)}^{\alpha_i} = \mathcal{TT}_0^o$. Moreover, the condition $M(0) + \sum_{i=1}^s \mathsf{Mk}(C_i)(0) = 1$ implies that the triple containers $C_1, \ldots, C_s$ can contain altogether only at most one type triple (we have $\mathsf{Mk}(\hat{\rho}_0^{all})(0) = 1$, so every type triple in some C_i adds one to the sum $\sum_{i=1}^s \mathsf{Mk}(C_i)(0)$). It follows that, while storing the type $C_1 \to \ldots \to C_s \to o$, we only need to remember which one of $C_1, \ldots, C_s$ is nonempty (or that all of them are empty). Beside of that, we have $|\Sigma| + 1$ possibilities for the flag set $F \in \mathcal{F}_k \subseteq \mathcal{F}_1$ (recall that $k \leq n = 1$): either F is empty, or it contains a $(1, a)$-flag for some $a \in \Sigma$ (and no other flags of order 1, hence no other flags at all). We also have $|\Sigma| + 1$ possibilities for $M \in \mathcal{M}_k \subseteq \mathcal{M}_1$, since $M(0)$ is determined by $C_1, \ldots, C_s$ due to the equation $M(0) + \sum_{i=1}^s \mathsf{Mk}(C_i)(0) = 1$, and $M(1) \in \{0, \ldots, |\Sigma|\}$. Thus, taking into account the inequality $s < |\alpha| \leq 2 \cdot |\mathcal{G}|$ (cf. Lemma 9.5), for k and α maximizing $\eta_1^{\mathcal{G}}$ we obtain that

$$\eta_1^{\mathcal{G}} = |\mathcal{TT}_k^\alpha| \leq (|\Sigma| + 1) \cdot (|\Sigma| + 1) \cdot (1 + s) \leq (|\Sigma| + 1)^2 \cdot 2 \cdot |\mathcal{G}|\,.$$

  In particular, $\eta_1^{\mathcal{G}}$ is polynomial in $|\mathcal{G}|$ and $|\Sigma|$.
- Finally, let $n \geq 2$. In the flag set, for every order $l \in \{1, \ldots, k\}$ we either have no flags of order l, or we have an (l, a)-flag for some $a \in \Sigma$ (and no other flags of order l); this gives $(|\Sigma| + 1)^k$ possibilities. In the marker multiset, for every order $l \in \{1, \ldots, k\}$ the number of order-l markers is in $\{0, \ldots, |\Sigma|\}$ (and, as previously, the number of order-0 markers is determined by $C_1, \ldots, C_s$); this also gives $(|\Sigma| + 1)^k$ possibilities. Every triple container C_i is a function from $\mathcal{TT}_{ord(\alpha_i)}^{\alpha_i}$ to $\{0, \ldots, |\Sigma|\}$; this gives $(|\Sigma| + 1)^{|\mathcal{TT}_{ord(\alpha_i)}^{\alpha_i}|}$ possibilities. Because $ord(\alpha_i) \leq ord(\alpha) - 1 \leq n - 1$, and because α_i (as a subsort of α) belongs to $\mathcal{S}_{\mathcal{G}}$, we have $|\mathcal{TT}_{ord(\alpha_i)}^{\alpha_i}| \leq \eta_{n-1}^{\mathcal{G}}$. Since moreover

$s \leq |\alpha| \leq 2 \cdot |\mathcal{G}|$ (cf. Lemma 9.5) and $k \leq n$, for $n \geq 2$ we obtain that

$$\eta^{\mathcal{G}}_n = |\mathcal{TT}^{\alpha}_k| \leq (|\Sigma|+1)^k \cdot (|\Sigma|+1)^k \cdot \prod_{i=1}^{s} (|\Sigma|+1)^{|\mathcal{TT}^{\alpha_i}_{ord(\alpha_i)}|} \leq (|\Sigma|+1)^{2n+2\cdot|\mathcal{G}|\cdot\eta^{\mathcal{G}}_{n-1}}.$$

It follows that for every $n \geq 2$ the number $\eta^{\mathcal{G}}_n$ is at most $(n-1)$-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$. □

We now bound the size of $\mathcal{U}^{\mathcal{G}}_{/\sim}$, that is, the number of equivalence classes of useful type judgments $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ satisfying $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$.

Lemma 9.7. *Let $m \in \mathbb{N}$. The size of $\mathcal{U}^{\mathcal{G}}_{/\sim}$ over all schemes $\mathcal{G}$ of order at most $m+1$ is at most m-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$ (where by "0-fold exponential" we mean "polynomial").*

Proof: Consider a type judgment $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ in $\mathcal{U}^{\mathcal{G}}$. As already said, there are at most $|\mathcal{G}|$ choices for Q. The order of Q is at most $m+1$, and the sort of Q belongs to $\mathcal{S}_{\mathcal{G}}$, so there are at most $\eta^{\mathcal{G}}_{m+1}$ choices for $\hat{\tau}$; by Lemma 9.6 the number $\eta^{\mathcal{G}}_{m+1}$ is at most m-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$. We now bound the number of choices for the type environment Γ. Recall that we may have $\Gamma(x) \neq \mathbf{0}$ only for variables x that appear in $\Lambda(\mathcal{G})$, so there exist at most $|\mathcal{G}|$ of them. Moreover, every such a variable x is bound by some subterm $\lambda x.R$ in $\Lambda(\mathcal{G})$; since $ord(\lambda x.R) \leq m+1$, we have $ord(x) \leq m$. We now consider two cases, depending on m.

- Suppose first that $m \geq 1$. Consider a variable x^{α} that appears in $\Lambda(\mathcal{G})$. Its sort α belongs to $\mathcal{S}_{\mathcal{G}}$, so we obtain $|\mathcal{TT}^{\alpha}_{ord(\alpha)}| \leq \eta^{\mathcal{G}}_m$. The type environment Γ assigns to x^{α} a triple container from $\mathcal{TC}^{\alpha}$, that is, a function from $\mathcal{TT}^{\alpha}_{ord(\alpha)}$ to $\{0,\ldots,|\Sigma|\}$. The number of such functions is at most $(|\Sigma|+1)^{\eta^{\mathcal{G}}_m}$. By taking a product over all variables appearing in $\Lambda(\mathcal{G})$, we obtain that the number of possible type environments Γ is at most $(|\Sigma|+1)^{|\mathcal{G}|\cdot\eta^{\mathcal{G}}_m}$. This number is at most m-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$ (cf. Lemma 9.5).
- Suppose now that $m = 0$ (then the bound from the previous item is exponential, while we need a polynomial one). In this case all variables that appear in $\Lambda(\mathcal{G})$ are of order 0 (thus of sort o). As already noticed, the only type in $\mathcal{TT}^{o}_{ord(o)} = \mathcal{TT}^{o}_0$ is $\hat{\rho}^{all}_0$, and it satisfies $\mathsf{Mk}(\hat{\rho}^{all}_0)(0) = 1$. On the other hand $\mathsf{Mk}(\hat{\tau})(0) \leq 1$ (by the definition of a marker multiset). Since we only consider type judgments satisfying $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$, the whole Γ assigns at most one type triple $\hat{\rho}^{all}_0$, to at most one variable. We thus only need to remember to which variable it is assigned (or that it is not assigned to any variable). We have at most $1+|\mathcal{G}|$ possibilities (thus polynomially many).

Altogether, it follows that $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ is at most m-fold exponential in $|\mathcal{G}|$ and $|\Sigma|$. □

Notice that without loss of generality we can assume that $|\Sigma| \leq |\mathcal{G}|$, since in Σ we do not need letters not appearing in $\mathcal{G}$ (actually, we can even assume that $|\Sigma| = |A|+1$, since a single letter name can be used for all letters that are not in A). Thus, we can simply say that $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ is at most m-fold exponential in $|\mathcal{G}|$.

9.2 Pumpable derivations

In the second subsection, we argue that we are actually interested in finding an A-pumpable derivation. We shall see here derivations as trees, similarly as in Section 6.3. For a node v of a derivation tree, by c_v we denote the flag counter being part of the type judgment written in v. We start by the following lemma, saying that the flag counter cannot grow too much in a single place.

Lemma 9.8. *When m and Σ are fixed, there exists a constant C such that for every $a \in \Sigma$, for every closed lambda-term S of sort o, for every derivation tree t that derives $\varepsilon \vdash_m S : \hat{\rho}_m^{all} \triangleright d$ (for some d), and for every node u of t, if $c_u(a) \geq C$, then there exists a child v of u such that $c_v(a) \geq \frac{1}{C} \cdot c_u(a)$.*

Proof: As C we take a number such that

$$C > 2m+3 \qquad \text{and} \qquad C > (m+2) \cdot (1 + |\Sigma| \cdot (m+1)) \,.$$

Fix now some $a \in \Sigma$, and consider a type judgment $\Gamma \vdash_m R : (F, M, \tau) \triangleright c$ appearing in t, such that $c(a) \geq C$. We have several cases, depending on the rule used to derive this type judgment. This cannot be the (VAR) rule, since it requires that $c(a) = 0$, while we have $c(a) \geq C > 0$. If this is the (ND) rule or the (λ) rule, the flag counter in the unique premiss of the rule is also c, so we trivially have $c(a) \geq \frac{1}{C} \cdot c(a)$.

Suppose now that the considered type judgment is derived using the (CON1) rule, that is, that R is of the form $b\langle P \rangle$, where $b \neq \mathsf{nd}$. Let $\Gamma \vdash_m P : (F_1, M, o) \triangleright c_1$ be the premiss of this rule. We have that $(F, c) \in \mathit{Comp}_m(M; (\{(1,b)\}, \mathbf{0}), (F_1, c_1))$. Consider the numbers $f_{k,a}$ and $f'_{k,a}$ as in the definition of the Comp_m predicate. We have that $f_{k,a} = f'_{k,a} + |\{(1,b)\} \cap \{(k,a)\}| + |F_1 \cap \{(k,a)\}| \leq f'_{k,a} + 2$ for $k \in \{1, \ldots, m+1\}$, and $f'_{k,a} \leq f_{k-1,a}$ for $k \in \{2, \ldots, m+1\}$, and $f'_{1,a} = 0$. It follows that $f_{m+1,a} \leq 2 \cdot (m+1) \leq 2 \cdot (m+1) \cdot \frac{1}{C} \cdot c(a)$ (the latter inequality holds because $c(a) \geq C$). Suppose now, contrary to the thesis, that $c_1(a) < \frac{1}{C} \cdot c(a)$. Then we have that

$$c(a) = f_{m+1,a} + c_1(a) \leq 2 \cdot (m+1) \cdot \frac{1}{C} \cdot c(a) + \frac{1}{C} \cdot c(a) \,,$$

which gives

$$C \leq 2 \cdot (m+1) + 1 = 2m + 3 \,.$$

This contradicts the assumption that $C > 2m+3$, thus necessarily $c_1(a) \geq \frac{1}{C} \cdot c(a)$.

In the case of the (CON0) rule, when $R = b\langle\rangle$, we simply have that $(F, c) \in \mathit{Comp}_m(M; (\{(1,b)\}, \mathbf{0}))$, so also $(F, c) \in \mathit{Comp}_m(M; (\{(1,b)\}, \mathbf{0}), (\emptyset, \mathbf{0}))$. The analysis from the previous paragraph gives us that $0 = \mathbf{0}(a) \geq \frac{1}{C} \cdot c(a)$, which is a clear contradiction with the assumption $c(a) \geq C$.

It remains to consider the case of the (@) rule, when $R = P\,Q$. Premisses of this rule are $\Gamma_0 \vdash_m P : (F_0, M_0, \tau_0) \triangleright c_0$ and $\Gamma_i \vdash_m Q : (F_i, M_i, \tau_i) \triangleright c_i$ for $i \in I$, where we assume that $0 \notin I$. We have that $(F, c) \in \mathit{Comp}_m(M; (F_0, c_0), ((F_i{\restriction}_{>ord(Q)}, c_i))_{i \in I})$. Again, we consider the numbers $f_{k,a}$ and $f'_{k,a}$. Let l be the smallest positive natural number such that $M(k) > 0$ for all $k \in \{l, l+1, \ldots, m\}$ (when $M(m) = 0$, we simply take $l = m+1$). Then by definition we have $f'_{l,a} = 0$ (since either $l = 1$ or $M(l-1) = 0$) and $f'_{k,a} = f_{k-1,a}$ for all $k \in \{l+1, l+2, \ldots, m+1\}$. We now prove that for all $k \in \{l, l+1, \ldots, m+1\}$,

$$f_{k,a} \leq f'_{k,a} + 1 + |\Sigma| \cdot (m+1) \,. \tag{3}$$

To this end, we consider two cases. Suppose first that $l \leq k \leq ord(Q)$. Then $F_i{\restriction}_{>ord(Q)} \cap \{(k,a)\} = \emptyset$ for all $i \in I$, so we obtain Inequality (3):

$$\begin{aligned} f_{k,a} &= f'_{k,a} + |F_0 \cap \{(k,a)\}| + \sum_{i \in I} |F_i{\restriction}_{>ord(Q)} \cap \{(k,a)\}| \\ &\leq f'_{k,a} + 1 \leq f'_{k,a} + 1 + |\Sigma| \cdot (m+1) \,. \end{aligned}$$

Next, suppose that $\max(ord(Q)+1, l) \leq k \leq m+1$. Notice that every index $i \in I$ with $M_i \neq \mathbf{0}$ adds at least one to the sum $\sum_{i\in I}\sum_{j=0}^{\infty} M_i(j)$. This sum cannot be greater than $|\Sigma|\cdot(m+1)$, since $\sum_{i\in I} M_i \leq \sum_{i\in\{0\}\cup I} M_i = M \in \mathcal{M}_m$. It follows that there are at most $|\Sigma|\cdot(m+1)$ indices $i \in I$ for which $M_i \neq \mathbf{0}$. On the other hand, from Lemma 5.5 we know that the derivation is not wild; in particular the considered use of the (@) rule is not wild. This means that for all $i \in I$ with $M_i = \mathbf{0}$ we have $(k,a) \notin F_i{\restriction}_{>ord(Q)}$ (since $k \geq l$, we have $M(j) > 0$ for all $j \in \{k, k+1, \ldots, m\}$, so $(k,a) \in F_i{\restriction}_{>ord(Q)}$ implies wildness). Thus, Inequality (3) follows also in this case:

$$f_{k,a} = f'_{k,a} + |F_0 \cap \{(k,a)\}| + \sum_{i\in I} |F_i{\restriction}_{>ord(Q)} \cap \{(k,a)\}| \leq f'_{k,a} + 1 + |\Sigma|\cdot(m+1)\,.$$

Using Inequality (3) for all possible k, and the assumption that $c(a) \geq C$, we obtain that

$$f_{m+1,a} \leq (m-l+2)\cdot(1+|\Sigma|\cdot(m+1)) \leq (m+1)\cdot(1+|\Sigma|\cdot(m+1))\cdot\frac{1}{C}\cdot c(a)\,.$$

Suppose now, contrary to the thesis, that $c_i(a) < \frac{1}{C}\cdot c(a)$ for all $i \in \{0\}\cup I$. For every $i \in I$ such that $M_i = \mathbf{0}$, by Lemma 5.4 (which can be applied because $ord(Q) \leq m$) we actually have that $c_i(a) = 0$. There are at most $|\Sigma|\cdot(m+1)$ indices $i \in I$ for which $M_i \neq \mathbf{0}$, that is, for which $c_i(a)$ can be nonzero (plus one more index $i = 0$). We thus obtain:

$$\begin{aligned} c(a) &= f_{m+1,a} + \sum_{i\in\{0\}\cup I} c_i(a) \\ &\leq (m+1)\cdot(1+|\Sigma|\cdot(m+1))\cdot\frac{1}{C}\cdot c(a) + (1+|\Sigma|\cdot(m+1))\cdot\frac{1}{C}\cdot c(a)\,, \end{aligned}$$

which gives

$$C \leq (m+2)\cdot(1+|\Sigma|\cdot(m+1))\,.$$

This contradicts the assumption that $C > (m+2)\cdot(1+|\Sigma|\cdot(m+1))$, thus necessarily $c_i(a) \geq \frac{1}{C}\cdot c(a)$ for some $i \in \{0\}\cup I$. □

In the next lemma we argue that it is enough to consider A-pumpable derivations.

Lemma 9.9. *Let S be a closed lambda-term of sort o. There exists an A-pumpable derivation of $\varepsilon \vdash_m S : \hat{\rho}^{all}_m \triangleright c$ for some flag counter c if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m S : \hat{\rho}^{all}_m \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$.*

Proof: Let us first justify the left-to-right implication. Suppose that a derivation tree t that derives $\varepsilon \vdash_m S : \hat{\rho}^{all}_m \triangleright c$ is A-pumpable. By definition, this means that for every letter $a \in A$, there are two nodes u_a, v_a of t such that u_a is an ancestor of v_a, they contain equivalent type judgments, and $c_{u_a}(a) \neq c_{v_a}(a)$. Our type system has the property that the flag counter in the conclusion of a rule is always not smaller than the flag counter in all the premisses, which implies that $c_{u_a}(a) > c_{v_a}(a)$, and $c_{u_a}(b) \geq c_{v_a}(b)$ for all $b \in \Sigma$. In the considered situation, for every $a \in A$ we can repeat (as many times as we want) the fragment of the derivation tree between the nodes u_a and v_a, increasing arbitrarily the a coordinate of the flag counter in the resulting type judgment. This is possible thanks to the following additivity property of

our type system: if out of $\Gamma \vdash_m P : \hat{\sigma} \triangleright c$ we can derive $\Gamma' \vdash_m P' : \hat{\sigma}' \triangleright c'$, then out of $\Gamma \vdash_m P : \hat{\sigma} \triangleright d$ we can derive $\Gamma' \vdash_m P' : \hat{\sigma}' \triangleright c' + d - c$.

We now prove the right-to-left implication. Let C be the constant from Lemma 9.8, and let $K = |\mathcal{U}^{\mathcal{G}}_{/\sim}|$. Take $n = C^K$, and consider a derivation tree t that derives $\varepsilon \vdash_m S : \hat{\rho}^{all}_m \triangleright c_n$ for some c_n such that $c_n(a) \geq n$ for all $a \in A$; it exists by assumption. We claim that t is A-pumpable. In order to prove this, take some letter $a \in A$. As already mentioned, $c_w(a) \leq c_u(a)$ whenever w is a child of u in t. We now construct a sequence $v_0, \ldots, v_K$ of $K+1$ nodes lying on one branch in t, such that $c_{v_i}(a) \geq C^{K-i}$ for $i \in \{0, \ldots, K\}$, and $c_{v_i}(a) < c_{v_{i-1}}(a)$ for $i \in \{1, \ldots, K\}$. As v_0 we take the root of t; then $c_{v_0}(a) = c_n(a) \geq C^K$. Next, suppose that $v_0, \ldots, v_{i-1}$ are already constructed, and we want to construct v_i (we do this by induction, for $i = 1, 2, \ldots, K$). Let u be some node in the subtree starting in v_{i-1}, such that $c_u(a) = c_{v_{i-1}}(a)$ but $c_w(a) < c_{v_{i-1}}(a)$ for all children w of u (such a node u has to exist, as t is finite). Then, as v_i we take a child of u such that $c_{v_i}(a) \geq \frac{1}{C} \cdot c_u(a)$, which exists by Lemma 9.8. Because $c_u(a) = c_{v_{i-1}}(a) \geq C^{K-i+1}$, it follows that $c_{v_i}(a) \geq C^{K-i}$; we also have $c_{v_i}(a) < c_{v_{i-1}}(a)$.

Once $v_0, \ldots, v_K$ are constructed, we notice that there is more of them than equivalence classes in $\mathcal{U}^{\mathcal{G}}_{/\sim}$. As already noticed, only type judgments from $\mathcal{U}^{\mathcal{G}}_{/\sim}$ may appear in t. It follows that among the nodes $v_0, \ldots, v_K$ there are two, v_i and v_j for $i < j$, labeled by equivalent type judgments. By construction v_i and v_j are located on the same branch, and we have that $c_{v_i}(a) > c_{v_j}(a)$. Such a pair of nodes can be found for every $a \in A$, so t is A-pumpable. □

9.3 Algorithms

We now give an algorithm that checks whether an A-pumpable derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}^{all}_m \triangleright c$ exists for some c; in consequence, it solves SUP for schemes. Recall that Parys (2017b) proposes an approach that is doubly exponential in $\mathcal{U}^{\mathcal{G}}_{/\sim}$: list all type derivations of height smaller than some number, and search among them for an A-pumpable derivation. We show how to search for A-pumpable derivations in a more systematic way.

Actually, we present two algorithms. Our first algorithm is deterministic, and works in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + |\mathcal{G}| + f(|A|)$ for some exponential function f (notice that when $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ is exponential in $|\mathcal{G}|$, the component $f(|A|)$ is anyway dominated by $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$). The second algorithm is nondeterministic, and works in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + |\mathcal{G}|$, so it avoids the exponential dependence on $|A|$. The existence of these algorithms proves the upper bounds required by Point 2 of Theorem 9.1, once we recall the bound on $\mathcal{U}^{\mathcal{G}}_{/\sim}$ from Lemma 9.7.

A type judgment is called *basic* if its flag counter is $\mathbf{0}$. Basic type judgments can be used to represent equivalence classes of type judgments, as in every equivalence class there is exactly one basic type judgment.

We denote type judgments using letters J, K, and L (possibly with some subscripts or superscripts). While denoting basic type judgments, we put 0 in the superscript, like in J^0. For a type judgment J, let $J{\downarrow}$ be the basic type judgment equivalent to J, and let c_J be the flag counter appearing in J.

Recall that while using the (@) rule, it may be needed to repeat the same premiss more than once. Thus, we should think about a *multiset* of premisses of a rule. In practice, it is more convenient to write a list of premisses, listing elements of the multiset in some particular order, which is arbitrary but fixed (the list is commutative: we can reorder premisses on the list, and the use of the rule will remain valid).

Our algorithm computes several sets, which we now define; their desired properties are described by Lemma 9.10. The set $\mathcal{D}$ (containing basic type judgments from $\mathcal{U}^{\mathcal{G}}$) is the smallest set such that:

- if by applying a rule to type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$ one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, then $J{\downarrow} \in \mathcal{D}$.

In the above definition we allow any $r \geq 0$ (in particular, we also consider rules that do not need any premiss). The set $\mathcal{E}$ (being a subset of $\mathcal{D} \times \mathcal{D}$) is the smallest set such that:

- $(J^0, J^0) \in \mathcal{E}$ for all $J^0 \in \mathcal{D}$, and
- if by applying a rule to type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$, where $r \geq 1$, one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, and $(J_1^0, K^0) \in \mathcal{E}$, then $(J{\downarrow}, K^0) \in \mathcal{E}$.

For every $a \in A$, the set $\mathcal{D}_a$ (being a subset of $\mathcal{D}$) is the smallest set such that:

- if by applying a rule to type judgments $J_1^0 \in \mathcal{D}_a$ and $J_2^0, \ldots, J_r^0 \in \mathcal{D}$ (where $r \geq 1$) one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, then $J{\downarrow} \in \mathcal{D}_a$, and
- if by applying a rule to type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$ one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$ satisfying $c_J(a) > 0$, then $J{\downarrow} \in \mathcal{D}_a$.

Finally, for every $a \in A$, the set $\mathcal{E}_a$ (being a subset of $\mathcal{D}_a \times \mathcal{D}$) is the smallest set such that:

- if by applying a rule to type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$, where $r \geq 1$, one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, and $(J_1^0, K^0) \in \mathcal{E}_a$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$,
- if by applying a rule to type judgments $J_1^0 \in \mathcal{D}_a$ and $J_2^0, \ldots, J_r^0 \in \mathcal{D}$, where $r \geq 2$, one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, and $(J_2^0, K^0) \in \mathcal{E}$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$, and
- if by applying a rule to type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$, where $r \geq 1$, one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$ satisfying $c_J(a) > 0$, and $(J_1^0, K^0) \in \mathcal{E}$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$.

Lemma 9.10. *In the setting as above:*

(a) the set $\mathcal{D}$ consists of projections $J{\downarrow}$ of all type judgments $J \in \mathcal{U}^{\mathcal{G}}$ that can be derived,

(b) the set $\mathcal{E}$ consists of pairs $(J{\downarrow}, K{\downarrow})$ for all type judgments $J \in \mathcal{U}^{\mathcal{G}}$ that can be derived so that K appears in a derivation of J,

(c) for every $a \in A$, the set $\mathcal{D}_a$ consists of projections $J{\downarrow}$ of all type judgments $J \in \mathcal{U}^{\mathcal{G}}$ that can be derived and that satisfy $c_J(a) > 0$, and

(d) for every $a \in A$, the set $\mathcal{E}_a$ consists of pairs $(J{\downarrow}, K{\downarrow})$ for all type judgments $J \in \mathcal{U}^{\mathcal{G}}$ that can be derived so that K appears in a derivation of J, and where $c_J(a) > c_K(a)$.

Proof: We first argue that every element in $\mathcal{D}$ (in $\mathcal{D}_a$) is of the form $J{\downarrow}$ for some type judgment $J \in \mathcal{U}^{\mathcal{G}}$ that can be derived (and satisfies $c_J(a) > 0$, respectively). This is shown by induction on the order in which type judgments are added to the set $\mathcal{D}$ (or $\mathcal{D}_a$) in its definition. By definition, some $J'{\downarrow}$ is added to $\mathcal{D}$, if it can be derived by applying some rule to basic type judgments $J_1^0, \ldots, J_r^0 \in \mathcal{D}$, where from the induction assumption we know for $i \in \{1, \ldots, r\}$ that $J_i^0 = J_i{\downarrow}$ for some type judgment $J_i \in \mathcal{U}^{\mathcal{G}}$ that can be derived. The same rule can be applied to the type judgments $J_1, \ldots, J_r$, and results in a type judgment J equivalent to J', where $c_J = c_{J'} + c_{J_1} + \cdots + c_{J_r}$. If $J_1^0 \in \mathcal{D}_a$ (the first item in the definition of $\mathcal{D}_a$), by the induction assumption we actually know that $c_{J_1}(a) > 0$, so also $c_J(a) > 0$. If $c_{J'}(a) > 0$ (the second item in the definition of $\mathcal{D}_a$) then automatically also $c_J(a) > 0$.

The opposite inclusion is shown by induction on the size of a fixed derivation of the considered derivable type judgment $J \in \mathcal{U}^{\mathcal{G}}$. Consider the final rule used in the derivation; let $J_1, \ldots, J_r$ be its premisses. As already said, $J_1, \ldots, J_r$ necessarily belong to $\mathcal{U}^{\mathcal{G}}$. By the induction assumption we have that $J_1{\downarrow}, \ldots, J_r{\downarrow} \in \mathcal{D}$. The application of the final rule remains valid if we replace the flag counters in $J_1, \ldots, J_r$ by $\mathbf{0}$, and we appropriately decrease the flag counter in J. This proves that $J{\downarrow} \in \mathcal{D}$.

When additionally $c_J(a) > 0$, and we want to prove that $J{\downarrow} \in \mathcal{D}_a$, we have two cases.

- Suppose first that $c_{J_k}(a) > 0$ for some $k \in \{1, \ldots, r\}$. By reordering premisses of the considered rule, we can assume that $k = 1$. We have $J_1{\downarrow} \in \mathcal{D}_a$ by the induction assumption, so $J{\downarrow} \in \mathcal{D}_a$ according to the first item in the definition of $\mathcal{D}_a$.
- Otherwise, $c_{J_i}(a) = 0$ for all $i \in \{1, \ldots, r\}$. Then, while replacing the flag counters in $J_1, \ldots, J_r$ by $\mathbf{0}$, we do not change the a coordinate of the flag counter in J, so it remains positive. Thus $J{\downarrow} \in \mathcal{D}_a$ according to the second item in the definition of $\mathcal{D}_a$.

The argumentation for the sets $\mathcal{E}$ and $\mathcal{E}_a$ is very similar, and thus it is left to the reader. □

We now come to pumpable derivations. Here we need one more definition: for a nonempty subset B of A we define a B-skeleton (we use B-skeletons to describe a general shape of B-pumpable derivations). For $B = \{a\}$ we have only one B-skeleton, which is a. For B of size at least 2, a B-skeleton is of the form either:

- $a[S]$, where S is a $(B \setminus \{a\})$-skeleton, or
- $(S_1), \ldots, (S_s)$, where S_i is a B_i-skeleton for $i \in \{1, \ldots, s\}$, for some partition of B into disjoint nonempty subsets $B_1, \ldots, B_s$, where $s \geq 2$.

Example $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$-skeletons are $\mathsf{a}[\mathsf{b}[\mathsf{c}]]$, and $\mathsf{c}[(\mathsf{b}), (\mathsf{a})]$, and (b), (a), (c), and (a), $((\mathsf{b}), (\mathsf{c}))$. It should be clear that an A-skeleton can be represented in a space polynomial in $|A|$, so the number of A-skeletons is at most exponential in $|A|$.

For every skeleton S we define a set $\mathcal{P}_S$ as the smallest set such that

- if by applying a rule to type judgments $J_1^0 \in \mathcal{P}_S$ and $J_2^0, \ldots, J_r^0 \in \mathcal{D}$ (where $r \geq 1$) one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, then $J{\downarrow} \in \mathcal{P}_S$,
- if S equals a, and $(J^0, J^0) \in \mathcal{E}_a$, then $J^0 \in \mathcal{P}_S$,
- if S equals $a[S']$, and $(J^0, J^0) \in \mathcal{E}_a$, and $J^0 \in \mathcal{P}_{S'}$, then $J^0 \in \mathcal{P}_S$, and
- if S equals $(S_1), \ldots, (S_s)$, and by applying a rule to type judgments $J_1^0 \in \mathcal{P}_{S_1}, \ldots, J_s^0 \in \mathcal{P}_{S_s}$ and $J_{s+1}^0, J_{s+2}^0, \ldots, J_r^0 \in \mathcal{D}$ (where $r \geq s$) one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, then $J{\downarrow} \in \mathcal{P}_S$.

Lemma 9.11. *Let $J^0 \in \mathcal{U}^{\mathcal{G}}$ be a basic type judgment, and let $B \subseteq A$ be nonempty. Then there exists a B-pumpable derivation of a type judgment equivalent to J^0 if and only if $J^0 \in \mathcal{P}_S$ for some B-skeleton S.*

Proof: We first suppose that $J^0 \in \mathcal{P}_S$ for some B-skeleton S, and we show a B-pumpable derivation of a type judgment equivalent to J^0. This is an induction on the size of B, and internally on the order in which type judgments are added to $\mathcal{P}_S$. We have several cases:

- Suppose that by applying a rule to type judgments $J_1^0 \in \mathcal{P}_S$ and $J_2^0, \ldots, J_r^0 \in \mathcal{D}$ one can derive a type judgment equivalent to J^0 (the first item in the definition). Then by the induction assumption we have a B-pumpable derivation of a type judgment equivalent to J_1^0, and for each $i \in \{2, \ldots, r\}$ by Lemma 9.10(a) we have a derivation of a type judgment equivalent to J_i^0. We finish the derivation by applying the considered rule, and we obtain a B-pumpable derivation of a type judgment equivalent to J^0.
- Suppose that S equals a, and $(J^0, J^0) \in \mathcal{E}_a$ (the second item in the definition). In this case $B = \{a\}$. Lemma 9.10(d) implies that there is a derivation of a type judgment J_1 equivalent to J^0 in which some J_2 equivalent to J^0 appears, where $c_{J_1}(a) > c_{J_2}(a)$. By definition such a derivation is B-pumpable.
- Suppose that S equals $a[S']$, and $(J^0, J^0) \in \mathcal{E}_a$, and $J^0 \in \mathcal{P}_{S'}$ (the third item in the definition). Recall that S' is a B'-skeleton for $B' = B \setminus \{a\}$. Lemma 9.10(d) implies that there is a derivation

tree t deriving a type judgments J_1 equivalent to J^0 in which some J_2 equivalent to J^0 appears, where $c_{J_1}(a) > c_{J_2}(a)$. Moreover, the induction assumption implies that there is a B'-pumpable derivation tree t' deriving a type judgment J_3 equivalent to J^0. We now insert t' in a node of t in which J_2 was written (cutting off all children of that node), and we modify appropriately flag counters on the path from this node to the root of t. This way, we obtain a B-pumpable derivation of a type judgment equivalent to J^0.

- Finally, suppose that S equals $(S_1), \ldots, (S_s)$, and by applying a rule to type judgments $J^0_1 \in \mathcal{P}_{S_1}, \ldots, J^0_s \in \mathcal{P}_{S_s}$ and $J^0_{s+1}, J^0_{s+2}, \ldots, J^0_r \in \mathcal{D}$ (where $r \geq s$) one can derive a type judgment equivalent to J^0 (the fourth item in the definition). Then $B = B_1 \cup \cdots \cup B_s$, where S_i is a B_i-skeleton for $i \in \{1, \ldots, s\}$. For $i \in \{1, \ldots, s\}$, by the induction assumption we have a B_i-pumpable derivation of a type judgment equivalent to J^0_i. Moreover, for $i \in \{s+1, s+2, \ldots, r\}$ by Lemma 9.10(a) we have a derivation of a type judgment equivalent to J^0_i. We finish the derivation by applying the considered rule, and we obtain a B-pumpable derivation of a type judgment equivalent to J^0.

Next, we prove the opposite implication. Consider thus a B-pumpable derivation of a type judgment J equivalent to J^0. We proceed by induction on the size of this derivation. Let $J_1, \ldots, J_r$ be the premisses of the final rule used in this derivation. We have several possibilities here:

- It is possible that already a subderivation resulting in J_k for some $k \in \{1, \ldots, r\}$ is B-pumpable. By reordering premisses, we can assume that $k = 1$. By the induction assumption, $J_1{\downarrow} \in \mathcal{P}_S$ for some B-skeleton S. Moreover, $J_2{\downarrow}, \ldots, J_r{\downarrow} \in \mathcal{D}$ by Lemma 9.10(a). By scaling down flag counters in the rule used in the root of the derivation, we obtain a situation as in the first item of the definition, so $J^0 \in \mathcal{P}_S$.
- Suppose now that a type judgment J' equivalent to J^0 appears somewhere in D, with $c_J(a) > c_{J'}(a)$ for some $a \in B$. Then $(J^0, J^0) \in \mathcal{E}_a$ by Lemma 9.10(d). If $B = \{a\}$, as S we take a, and we obtain $J^0 \in \mathcal{P}_S$ by the second item of the definition. Suppose thus that $|B| \geq 2$. Let $B' = B \setminus \{a\}$. A B-pumpable derivation is also B'-pumpable, so by the induction assumption we have that $J^0 \in \mathcal{P}_{S'}$ for some B'-skeleton S'. As S we take $a[S']$, and then $J^0 \in \mathcal{P}_S$ by the third item of the definition.
- Finally, suppose that the two above possibilities do not hold. Then necessarily there is a subsequence $J_{j_1}, \ldots, J_{j_s}$ of $J_1, \ldots, J_r$, and a partition of B into disjoint nonempty subsets $B_1, \ldots, B_s$ (where $s \geq 2$) such that for every $i \in \{1, \ldots, s\}$ the subderivation resulting in J_{j_i} is B_i-pumpable. By reordering premisses, we can assume that $(j_1, \ldots, j_s) = (1, \ldots, s)$. For every $i \in \{1, \ldots, s\}$ by the induction assumption we have that $J_i{\downarrow} \in \mathcal{P}_{S_i}$ for some B_i-skeleton S_i. Moreover, $J_{s+1}{\downarrow}, J_{s+2}{\downarrow}, \ldots, J_r{\downarrow} \in \mathcal{D}$ by Lemma 9.10(a). By scaling down flag counters in the rule used in the root of the derivation, we obtain a situation as in the fourth item of the definition, so $J^0 \in \mathcal{P}_S$. □

Finally, let us see that all the sets can be quickly computed. We start by a lemma describing a single rule.

Lemma 9.12. *Given a set of basic type judgments $\mathcal{D} \subseteq \mathcal{U}^{\mathcal{G}}$, and its subsets $\mathcal{D}_1, \ldots, \mathcal{D}_s \subseteq \mathcal{D}$, and a basic type judgment $J^0 \in \mathcal{U}^{\mathcal{G}}$, and a letter $a_+ \in \Sigma$, it can be decided in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + s$ whether there exist type judgments $J^0_1 \in \mathcal{D}_1, \ldots, J^0_s \in \mathcal{D}_s$ and $J^0_{s+1}, J^0_{s+2}, \ldots, J^0_r \in \mathcal{D}$ (for some $r \geq s$) such that by applying a type system rule to $J^0_1, \ldots, J^0_r$ one can derive:*
(a) a type judgment J that is equivalent to J^0;
(b) a type judgment J that is equivalent to J^0 and such that $c_J(a_+) > 0$.

Proof: This lemma is not completely obvious, as the number r of premisses can be arbitrarily large (the

same type judgment can be even repeated in the list of premisses), so we cannot iterate through all possible lists of type judgments from $\mathcal{D}$. But let analyze every rule separately.

The rules (VAR) and (CON0) have no premisses, so they require $s = 0$. It can be easily checked whether some J equivalent to J^0 can be derived, and whether its flag counter c_J can satisfy $c_J(a_+) > 0$.

For the rules (ND), (λ), and (CON1) the situation is also easy, as they require exactly one premiss. We can thus loop over all type judgments $J_1^0 \in \mathcal{D}$. For $s = 1$ we require that $J_1^0 \in \mathcal{D}_1$, and for $s \geq 2$ we always fail. When the premiss and the conclusion are fixed (modulo the value of the flag counter in the conclusion), it is straightforward to check whether the rule can be applied, and whether the flag counter c in the conclusion can satisfy $c(a_+) > 0$.

Consider now the (@) rule. It is useful to define a predicate $Comp'_m(k, a, M, F')$ which is true if $k \in \{1, \dots, m+1\}$, and $a \in \Sigma$, and there exists $l \in \{1, \dots, k\}$ such that $(l, a) \in F'$ and $M(i) > 0$ for all i satisfying $l \leq i \leq k-1$. It follows directly from the definition of $Comp_m$ that for any $M, F_1, \dots, F_n$ the set

$$\{F \mid (F, c) \in Comp_m(M; (F_1, \mathbf{0}), \dots, (F_n, \mathbf{0})) \text{ for some } c\}$$

contains exactly these sets F for which

$$\forall (k, a) \in F \,.\, \exists i \in \{1, \dots, n\} \,.\, Comp'_m(k, a, M, F_i)\,.$$

Moreover, the set

$$\{F \mid (F, c) \in Comp_m(M; (F_1, \mathbf{0}), \dots, (F_n, \mathbf{0})) \text{ for some } c \text{ with } c(a_+) > 0\}$$

contains exactly these sets F for which

$$\forall (k, a) \in F \cup \{(m+1, a_+)\} \,.\, \exists i \in \{1, \dots, n\} \,.\, Comp'_m(k, a, M, F_i)\,.$$

Suppose that $\Gamma \vdash_m P\, Q : (F, M, \tau) \triangleright c$ is the considered conclusion (where c is not fixed). We need to have one premiss concerning P, and an arbitrary number of premisses concerning Q, but the key point is that we do not need to know all premisses simultaneously. Having some number of premisses concerning Q, namely $\Gamma_i \vdash_m Q : (F_i, M_i, \tau_i) \triangleright \mathbf{0}$ for $i \in \{1, \dots, n\}$, and a premiss $\Gamma_0 \vdash_m P : (F_0, M_0, C \to \tau) \triangleright \mathbf{0}$ concerning P, we only need to remember

- the premiss J_P concerning P, or information saying that this premiss is not yet selected,
- the union $\Gamma' = \Gamma_1 \sqcup \dots \sqcup \Gamma_n$,
- the sum $M' = M_1 + \dots + M_n$,
- the set F' of these $(k, a) \in F \cup \{(m+1, a_+)\}$ for which $Comp'_m(k, a, M, F_i{\restriction}_{>ord(Q)})$ is satisfied for some $i \in \{1, \dots, n\}$,
- the triple container $C' = \{\!\{(F_i{\restriction}_{\leq ord(Q)}, M_i{\restriction}_{\leq ord(Q)}, \tau_i) \mid i \in \{1, \dots, n\}\}\!\}$, and
- the number $s' < s$ of selected premisses, or information that we have already selected s or more premisses.

These tuples $(J_P, \Gamma', M', F', C', s')$ satisfy $\Gamma' \leq \Gamma$, and $M' \leq M$, and $F' \subseteq F \cup \{(m+1, a_+)\}$, and $C' \leq C$, and $s' \leq s$. The number of such tuples is at most $2 \cdot (|\mathcal{U}^{\mathcal{G}}_{/\sim}| + 1) \cdot |\mathcal{U}^{\mathcal{G}}_{/\sim}|^4 \cdot (s+1)$, because all possible choices for Γ' (and similarly for M', for $F' \cap F$, for C', and for J_P) can appear in some type judgments from $\mathcal{U}^{\mathcal{G}}$.

In the algorithm we make a saturation loop, storing the set of already discovered tuples $(J_P, \Gamma', M', F', C', s')$ that can be obtained after selecting some number of premisses. We begin with the tuple $(-, \varepsilon, \mathbf{0}, \emptyset, \mathbf{0}, 0)$ corresponding to the empty list of premisses. Then, we take a tuple $(J_P, \Gamma', M', F', C', s')$ from the set (the tuple corresponds to an imaginary list of premisses), and we take a candidate for the next premiss, which should be from $\mathcal{D}_{s'+1}$ if $s' < s$, and from $\mathcal{D}$ otherwise. Knowing the tuple and the premiss, we can easily compute the tuple obtained after including this premiss; we add the computed tuple to the set of discovered tuples. When the set is computed (i.e., no more new tuples can be added), we determine for each tuple in the set whether a list of premisses described by this tuple allows to derive the desired conclusion. Such an algorithm is polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + s$ (recall that each of the sets $\mathcal{D}, \mathcal{D}_1, \ldots, \mathcal{D}_s$ is of size at most $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$). □

Having established Lemma 9.12, we come back to the main algorithm.

Lemma 9.13. *Let $m \in \mathbb{N}$. If $m \geq 1$, then there is an m-EXPTIME algorithm that given a scheme $\mathcal{G}$ of order at most $m+1$, a set $A \subseteq \Sigma$, a subterm R^α of $\Lambda(\mathcal{G})$, and a type triple $\hat{\tau} \in \mathcal{TT}^\alpha_m$, checks whether there exists an A-pumpable derivation of $\varepsilon \vdash_m R : \hat{\tau} \triangleright c$ for some flag counter c. If $m = 0$, the same can be done in NP, and in FPT when $|A|$ is viewed as a parameter.*

Proof: We need to compute all the sets described above. Let us start with the set $\mathcal{D}$. It can be computed by a saturation algorithm, following its definition. We start by taking $\mathcal{D} = \emptyset$. Then, in a loop, we check for every basic type judgment $J^0 \in \mathcal{U}^{\mathcal{G}}$ whether some type judgment J equivalent to J^0 can be obtained by applying some rule to some type judgments $J^0_1, \ldots, J^0_r$ belonging to the current version of $\mathcal{D}$; if so, we add J^0 to $\mathcal{D}$. Every such check can be done quickly due to Lemma 9.12(a) (where we take $s = 0$). We enlarge the set $\mathcal{D}$ at most $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ times, and after every change of $\mathcal{D}$ we need to check at most $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ basic type judgments. Overall, the computation works in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$.

The sets $\mathcal{D}_a$ can be computed similarly. Here, we need to check whether some type judgment J equivalent to J^0 can be obtained by applying some rule to some type judgments $J^0_2, \ldots, J^0_r$ belonging to the set $\mathcal{D}$, and J^0_1 belonging to the current version of $\mathcal{D}_a$ (the first item of the definition); this can be done by Lemma 9.12(a), where as $\mathcal{D}_1$ we take $\mathcal{D}_a$, and $s = 1$. We also need to check whether some type judgment J equivalent to J^0 and with flag counter satisfying $c_J(a) > 0$ can be obtained by applying some rule to some type judgments $J^0_1, \ldots, J^0_r$ belonging to the set $\mathcal{D}$ (the second item of the definition); this can be done by Lemma 9.12(b) (where $s = 0$ and $a_+ = a$).

The set $\mathcal{E}$ is also computed by a saturation algorithm. Here we loop over triples of basic type judgments J^0, K^0, L^0 such that $(L^0, K^0) \in \mathcal{E}$, and in a simple check we fire Lemma 9.12(a) with $\mathcal{D}_1 = \{L^0\}$ and $s = 1$. While computing sets $\mathcal{E}_a$ we have three kinds of checks, but again all of them can be handled by Lemma 9.12, where $s \leq 2$.

Finally, we show how to compute the set $\mathcal{P}_S$ for a fixed skeleton S, assuming that we have already computed the set $\mathcal{P}_{S'}$ if $S = a[S']$, and the sets $\mathcal{P}_{S_1}, \ldots, \mathcal{P}_{S_s}$ if S equals $(S_1), \ldots, (S_s)$. Here we have four kinds of checks, corresponding to the four items of the definition. The first of them is similar to what we did previously. The next two do not even require to use Lemma 9.12. The fourth item is more complicated, but Lemma 9.12(a) is perfectly suited to solve it.

Recall that our goal is to check whether there exists an A-pumpable derivation of $\varepsilon \vdash_m R : \hat{\tau} \triangleright c$ for some flag counter c. If $A = \emptyset$, every derivation is A-pumpable, so (due to Lemma 9.10(a)) it is enough to check whether $\varepsilon \vdash_m R : \hat{\tau} \triangleright \mathbf{0}$ belongs to $\mathcal{D}$. Suppose now that $A \neq \emptyset$. In this case, due to the equivalence given by Lemma 9.11, we need to check whether the type judgment $\varepsilon \vdash_m R : \hat{\tau} \triangleright \mathbf{0}$ belongs

to $\mathcal{P}_S$ for some A-skeleton S. Here is the only place where nondeterminism helps. If we can proceed nondeterministically, then we simply guess an A-skeleton S, we compute $\mathcal{P}_S$ only for this skeleton (and recursively for its subskeletons), and we check whether the type judgment belongs there. As already said, a single set $\mathcal{P}_S$ can be computed in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$. If we want to be deterministic, we compute the sets $\mathcal{P}_S$ for all A-skeletons S (their number is exponential in $|A|$), and we check whether the type judgment belongs to some of them. This finishes the proof once we recall from Lemma 9.7 that $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ is at most m-fold exponential (polynomial if $m = 0$) in $|\mathcal{G}|$ and $|\Sigma|$. □

From Theorem 3.2 we know that $SUP_A(\mathcal{L}(\mathcal{G}))$ holds for a word-recognizing scheme $\mathcal{G}$ of order at most $m+1$ if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}^{all}_m \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$; by Lemma 9.9 the latter holds if and only if there exists an A-pumpable derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}^{all}_m \triangleright c$ for some flag counter c. Thus, Lemma 9.13 implies all the upper bounds from Theorem 9.1 (recall that m is shifted by one with respect to the statement of this theorem).

9.4 Lower bounds

We now prove the lower bounds appearing in Theorem 9.1. We base here on the language nonemptiness problem for higher-order recursion schemes.

Lemma 9.14. *Let $m \geq 1$. It is m-EXPTIME-hard to decide, given a scheme $\mathcal{G}$ of order at most m, whether $\mathcal{L}(\mathcal{G})$ is nonempty.*

Proof: Kobayashi and Ong (2011, Corollary 3.7) prove that for $m \geq 1$ the following problem is m-EXPTIME-hard: given a scheme $\mathcal{G}$ of order at most m, and a trivial alternating parity tree automaton $\mathcal{B}$, decide whether $\mathcal{B}$ accepts $BT(\Lambda(\mathcal{G}))$. Instead of recalling the definition of a trivial alternating parity tree automaton, we notice that $\mathcal{G}$ and $\mathcal{B}$ produced by their reduction are of a special form. Namely, the tree $BT(\Lambda(\mathcal{G}))$ consists of n-ary nodes (for some $n \in \mathbb{N}$) labeled by A or E, and of leaves labeled by T or R. Let L be the smallest language such that:

- L contains the tree $\mathsf{T}\langle\rangle$,
- if L contains a tree T, then L contains every tree of the form $\mathsf{E}\langle T_1, \dots, T_n\rangle$ in which $T = T_i$ for some $i \in \{1, \dots, n\}$, and
- if L contains trees $T_1, \dots, T_n$, then L contains the tree $\mathsf{A}\langle T_1, \dots, T_n\rangle$.

In other words: if T and R are seen as "true" and "false", and A and E are seen as conjunction and disjunction, the language L contains expressions that evaluate to "true". The automaton $\mathcal{B}$ produced in the reduction is always the same (modulo the fact that n can vary), and it accepts those trees of the above form that do not belong to L. Let us rename all E and R letters appearing in $\mathcal{G}$ to nd; call the resulting scheme $\mathcal{G}'$. It is not difficult to see that $BT(\Lambda(\mathcal{G})) \in L$ (i.e., $\mathcal{B}$ rejects $BT(\Lambda(\mathcal{G}))$) if and only if $\mathcal{L}(\mathcal{G}')$ is nonempty. The m-EXPTIME complexity class is closed under taking the complement of a language. It follows that it is m-EXPTIME-hard to decide, given a scheme $\mathcal{G}'$ of order at most m, whether $\mathcal{L}(\mathcal{G}')$ is nonempty. □

A lower bound for SUP follows immediately.

Corollary 9.15. *Let $m \geq 1$. SUP for tree-recognizing order-m schemes is m-EXPTIME-hard.*

Proof: For the special case of $A = \emptyset$, SUP checks precisely whether the language is nonempty, that is, $SUP_\emptyset(\mathcal{L}(\mathcal{G}))$ holds if and only if $\mathcal{L}(\mathcal{G}) \neq \emptyset$. This way, the language nonemptiness problem for tree-recognizing order-m schemes reduces to SUP for tree-recognizing order-m schemes; thus m-EXPTIME-hardness of the former (cf. Lemma 9.14) implies m-EXPTIME-hardness of the latter. □

One may wonder whether the hardness result also holds when A is nonempty or, in particular, when A contains all letters appearing in the scheme. As expected, this is the case.

Corollary 9.16. *Let $m \geq 1$. It is m-EXPTIME-hard to decide, given a scheme $\mathcal{G}$ of order at most m and using only one letter* a *(besides the* nd *symbol), whether $SUP_{\{\mathsf{a}\}}(\mathcal{L}(\mathcal{G}))$ holds.*

Proof: Let us reduce the problem of nonemptiness of $\mathcal{L}(\mathcal{G})$ to the problem of deciding $SUP_{\{\mathsf{a}\}}(\mathcal{L}(\mathcal{G}'))$, where $\mathcal{G}'$ is a scheme using only one letter a. To produce $\mathcal{G}'$ we add to $\mathcal{G}$ a fresh starting nonterminal N_0' with a rule $\mathcal{R}(N_0') = \mathsf{nd}\langle N_0, \mathsf{a}\langle N_0'\rangle\rangle$, where N_0 is the starting nonterminal of $\mathcal{G}$. Moreover, we rename every letter appearing in $\mathcal{G}$, other than nd, to a. Notice that $\mathcal{G}'$ allows to precede every tree from $\mathcal{L}(\mathcal{G})$ by a sequence of any number of letters a; thus $SUP_{\{\mathsf{a}\}}(\mathcal{L}(\mathcal{G}'))$ holds if and only if $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Moreover, the orders of $\mathcal{G}$ and $\mathcal{G}'$ are the same. □

Recall from Section 8 that we can (in logarithmic space) translate a tree-recognizing scheme $\mathcal{G}$ of order at most m into a word-recognizing scheme $\mathcal{G}^\flat$ of order at most $m+1$. Thus, the above corollaries imply also m-EXPTIME-hardness of SUP for word-recognizing order-$(m+1)$ schemes (assuming $m \geq 1$).

In the last part, we prove that SUP for word-recognizing order-0 schemes is NP-hard. Of course this problem is a special case of SUP for word-recognizing order-1 schemes, and of SUP for tree-recognizing order-0 schemes, so the latter two problems are also NP-hard.

Lemma 9.17. *SUP for word-recognizing order-*0 *schemes is NP-hard.*

Proof: We reduce from the undirected Hamiltonian cycle problem, known to be NP-complete (Karp, 1972). We are thus given an undirected graph G, and we construct a word-recognizing scheme $\mathcal{G}$ of order 0 such that $SUP_A(\mathcal{G})$ holds if and only if there exists a Hamiltonian cycle in G. Suppose that the nodes of G are named $1, \ldots, s$, and $s \geq 2$. In $\mathcal{G}$ we use letters $\mathsf{a}_1, \ldots, \mathsf{a}_s$, and we take all of them to the set A. The nonterminals of $\mathcal{G}$ are N^i_j for $i \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, s\}$, all of sort o. For $i, j \in \{1, \ldots, s\}$ the rules are:

$$\mathcal{R}(\mathsf{N}^i_j) = \mathsf{a}_j\langle \mathsf{nd}\langle \mathsf{N}^i_j, \mathsf{N}^{i-1}_{v_1}, \ldots, \mathsf{N}^{i-1}_{v_r}\rangle\rangle\,,$$

where $v_1, \ldots, v_r$ are all neighbors of j in G. Moreover,

$$\begin{aligned} &\mathcal{R}(\mathsf{N}^0_1) = \mathsf{a}_1\langle\rangle, && \text{and} \\ &\mathcal{R}(\mathsf{N}^0_j) = \mathsf{nd}\langle\rangle && \text{for } j \in \{2, \ldots, s\}. \end{aligned}$$

As the starting nonterminal we take N^s_1.

By induction on $i \in \{0, \ldots, s\}$ we can see that $\mathcal{L}(BT(\Lambda_{\mathcal{G}}(\mathsf{N}^i_j)))$ contains words of the form

$$(\mathsf{a}_{j_i})^{n_i}(\mathsf{a}_{j_{i-1}})^{n_{i-1}} \ldots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0}\,,$$

where $n_1, \ldots, n_i$ are arbitrary positive numbers, and $j_i, j_{i-1}, \ldots, j_0$ is a path in G such that $j_i = j$ and $j_0 = 1$. It follows that $\mathcal{L}(\mathcal{G})$ contains words of the form $(\mathsf{a}_{j_s})^{n_s}(\mathsf{a}_{j_{s-1}})^{n_{s-1}} \ldots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0}$, where $n_1, \ldots, n_s$ are arbitrary positive numbers, and $j_s, j_{s-1}, \ldots, j_0$ is a path in G such that $j_s = j_0 = 1$. If G contains Hamiltonian cycles, as $j_s, j_{s-1}, \ldots, j_0$ we can take one of them, starting and ending in node 1 (by definition all Hamiltonian cycles have length s). Then the words $(\mathsf{a}_{j_s})^n(\mathsf{a}_{j_{s-1}})^n \ldots (\mathsf{a}_{j_1})^n\mathsf{a}_{j_0}$ for all $n \geq 1$ are in $\mathcal{L}(\mathcal{G})$ and contain every letter from $A = \{\mathsf{a}_1, \ldots, \mathsf{a}_s\}$ at least n times, so they

witness that $SUP_A(\mathcal{G})$ holds. Conversely, if G does not contain Hamiltonian cycles, then on every path $j_s, j_{s-1}, \dots, j_0$ with $j_s = j_0 = 1$ some node $k \in \{2, \dots, s\}$ is missing. In consequence, in every word $(\mathsf{a}_{j_s})^{n_s}(\mathsf{a}_{j_{s-1}})^{n_{s-1}} \dots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0} \in \mathcal{L}(\mathcal{G})$ some letter $\mathsf{a}_k \in A$ does not appear at all, so $SUP_A(\mathcal{G})$ does not hold. This proves that the reduction is correct. □

10 Reflection for SUP

In this section we establish the reflection property for SUP (Theorem 10.2). We recall that this property is a key ingredient used while proving decidability of model-checking trees generated by schemes with respect to the WMSO+U logic (Parys, 2018b).

The notion of reflection can be defined for an arbitrary property of trees. We describe the property by a relation Ξ between trees and letters.

Definition 10.1. Let Ξ be a relation between trees and letters. We define by coinduction when a tree T' is a Ξ-*reflection* of a tree T: if $T = a\langle T_1, \dots, T_r\rangle$, and $T' = b\langle T'_1, \dots, T'_r\rangle$, and $(T, b) \in \Xi$, and T'_i is a Ξ-reflection of T_i for every $i \in \{1, \dots, r\}$, then T' is a Ξ-reflection of T.

In other words, a Ξ-reflection of T has the same shape as T and is obtained by replacing the label of every node v in T by a letter that is related by Ξ to the subtree of T starting in v. Notice that if Ξ is a function, then every tree has a unique Ξ-reflection (while in general there may be multiple choices for the new labels).

We describe SUP using the function Ξ_{SUP} defined by $\Xi_{SUP}(T) = (a, \{A \subseteq \Sigma \mid SUP_A(\mathcal{L}(T))\})$ if $T = a\langle T_1, \dots, T_r\rangle \neq \mathsf{nd}\langle\rangle$, and $\Xi_{SUP}(\mathsf{nd}\langle\rangle) = \mathsf{nd}$. This implies that in the Ξ_{SUP}-reflection of T we decorate every node v by the collection of all sets A for which the answer to SUP for A and for the subtree of T starting in v is positive. For technical convenience we do not relabel nd-labeled leaves to $(\mathsf{nd}, \{\emptyset\})$, but rather we leave them nd-labeled (the reason is that some nd-labeled leaves in a Böhm tree are not created explicitly by a node constructor, but rather because some subterms do not reduce to terms starting with a node constructor, and we do not want to detect these subterms).

In Theorem 10.2 we claim that Ξ_{SUP}-reflections of trees generated by schemes are also generated by schemes.

Theorem 10.2. *Let $m \in \mathbb{N}$. For every scheme $\mathcal{G}$ of order at most m one can construct a scheme $\mathcal{G}_{SUP}$ of order at most m such that $BT(\Lambda(\mathcal{G}_{SUP}))$ is a Ξ_{SUP}-reflection of $BT(\Lambda(\mathcal{G}))$. Moreover, $\mathcal{G}_{SUP}$ can be constructed in time at most $\max(m, 1)$-fold exponential in $|\mathcal{G}|$, and doubly exponential in $|\Sigma|$; if $\mathcal{G}$ is word-recognizing, then $\mathcal{G}_{SUP}$ can be constructed in time at most $\max(m-1, 1)$-fold exponential in $|\mathcal{G}|$, and doubly exponential in $|\Sigma|$ (in both cases, this is simultaneously a bound on the size of $\mathcal{G}_{SUP}$).*

It can be assumed that $|\Sigma| \leq |\mathcal{G}|$, so the part saying that the running time is doubly exponential in $|\Sigma|$ is meaningful only when the complexity in $|\mathcal{G}|$ would be singly exponential (i.e., when $m \leq 1$ for general schemes, and when $m \leq 2$ for word-recognizing schemes).

Theorem 10.2 is obtained by instantiating (and adapting) a general construction of Haddad (2012, Section 4.2), which allows to obtain reflection for any property described by a morphism from closed lambda-terms to a finitary applicative structure. An *applicative structure* D consists of

- a set $D[\alpha]$ for each sort α,
- an *application* operation that to all elements $\chi \in D[\alpha \to \beta]$ and $\chi' \in D[\alpha]$ assigns an element $(\chi\,\chi') \in D[\beta]$, and

- an *abstraction* operation that to every element $\chi \in D[\beta]$ assigns an element $\boldsymbol{\lambda}\alpha.\chi \in D[\alpha \to \beta]$.[(vi)]

The two operations should be defined for every (homogeneous) sort of the form $\alpha \to \beta$. We say that D is *finitary* when $D[\alpha]$ is finite for every sort α. A *morphism* $\mathcal{M}$ to an applicative structure D is defined as a mapping that to each closed lambda-term P^α assigns an element of $D[\alpha]$ denoted $[\![P]\!]_{\mathcal{M}}$, such that

- for every closed lambda-term of the form $P\,Q$ it is the case that $[\![P]\!]_{\mathcal{M}}\,[\![Q]\!]_{\mathcal{M}} = [\![P\,Q]\!]_{\mathcal{M}}$, and
- for every lambda-term of the form $\lambda x^\alpha.P$, where P is closed, it is the case that $[\![\lambda x.P]\!]_{\mathcal{M}} = \boldsymbol{\lambda}\alpha.[\![P]\!]_{\mathcal{M}}$.

Let us emphasize that the second condition concerns only a situation where P is closed (i.e., where the variable x is not used is P).

For a morphism $\mathcal{M}$, and for a number $m \in \mathbb{N}$, we let $\Xi^m_{\mathcal{M}}$ to be the set of pairs $(BT(P), (a, [\![P]\!]_{\mathcal{M}}))$ over all closed lambda-terms P of sort o and of complexity at most m, where a is the label of the root of $BT(P)$, containing additionally the pair $(\mathsf{nd}\langle\rangle, \mathsf{nd})$. Notice that in a $\Xi^m_{\mathcal{M}}$-reflection we decorate every node v by $[\![P]\!]_{\mathcal{M}}$ for some closed lambda-term P of sort o and of complexity at most m such that $BT(P)$ equals the subtree starting in v. There are multiple choices for the lambda-term P, and in general $[\![P]\!]_{\mathcal{M}}$ may depend on the choice of P (but clearly at least one such P exists, namely P equal to the subtree starting in v itself; thus every tree has some $\Xi^m_{\mathcal{M}}$-reflection). The construction of Haddad can be summarized by the following lemma (recall from Section 9.1 that $\mathcal{S}_{\mathcal{G}}$ denotes the set of all sorts appearing in $\mathcal{G}$).

Lemma 10.3. *Let $m \in \mathbb{N}$. For every scheme $\mathcal{G}$ of order at most m, and for every morphism $\mathcal{M}$ to a finitary applicative structure D, one can construct a scheme $\mathcal{G}_{\mathcal{M}}$ of order at most m such that $BT(\Lambda(\mathcal{G}_{\mathcal{M}}))$ is a $\Xi^m_{\mathcal{M}}$-reflection of $BT(\Lambda(\mathcal{G}))$. Moreover,*

$$|\mathcal{G}_{\mathcal{M}}| \leq |\mathcal{G}| \cdot \left(\max_{\substack{\alpha \in \mathcal{S}_{\mathcal{G}} \\ ord(\alpha) \leq m-1}} |D[\alpha]| \right)^{|\mathcal{G}|^2},$$

and $\mathcal{G}_{\mathcal{M}}$ can be constructed in time polynomial in $|\mathcal{G}_{\mathcal{M}}|$ and in the time needed to

(A) compute the value $[\![\Lambda_{\mathcal{G}}(P)]\!]_{\mathcal{M}}$, given a closed lambda-term P of complexity at most m and of size polynomial in $|\mathcal{G}|$,

(B) compute the composition $\chi\,\chi'$, given two elements $\chi \in D[\alpha \to \beta], \chi' \in D[\alpha]$, where $\alpha \to \beta$ is of order at most m and of size polynomial in $|\mathcal{G}|$,

(C) compute the abstraction $\boldsymbol{\lambda}\alpha.\chi$, given a sort $\alpha \to \beta$ of order at most m and of size polynomial in $|\mathcal{G}|$, and an element $\chi \in D[\beta]$, and

(D) enumerate all elements of $D[\alpha]$ for $\alpha \in \mathcal{S}_{\mathcal{G}}$ such that $ord(\alpha) \leq m - 1$.

A construction of $\mathcal{G}_{\mathcal{M}}$ is presented in Haddad (2012, Section 4.2) for a particular morphism $\mathcal{M}$. In Haddad (2013b, Theorem 4) (with a proof in an unpublished appendix (Haddad, 2013a, Appendix A)) it is claimed to work for an arbitrary morphism that is invariant under beta-reductions, which means that if $P \to_\beta Q$ then necessarily $[\![P]\!]_{\mathcal{M}} = [\![Q]\!]_{\mathcal{M}}$. We remark that our morphisms (defined below) are not invariant under beta-reductions. Nevertheless, one can notice that the construction remains correct even without this assumption. Indeed, the only problem that may be caused by lack of invariance under

[(vi)] As one can see in Appendix B, the abstraction operation (and second condition in the definition of a morphism) is needed only because we assume that all sorts are homogeneous (in particular $[\![P]\!]_{\mathcal{M}}$ is defined only for lambda-terms of homogeneous sorts). If $[\![P]\!]_{\mathcal{M}}$ was defined also for lambda-terms of non-homogeneous sorts, an analogue of Lemma 10.3 would work already for structures without the abstraction operation.

beta-reductions is that some node, instead of being annotated by $[\![P]\!]_{\mathcal{M}}$ for some P, will be annotated by $[\![Q]\!]_{\mathcal{M}}$ for some Q that is beta-equivalent to P. This is not a problem, though, since for beta-equivalent lambda-terms we have $BT(P) = BT(Q)$, and thus if $(T, (a, [\![P]\!]_{\mathcal{M}}))$ is in $\Xi^m_{\mathcal{M}}$ then also $(T, (a, [\![Q]\!]_{\mathcal{M}}))$ is in $\Xi^m_{\mathcal{M}}$. For completeness, we give the construction and we justify its correctness in details in Appendix B.

We remark that Salvati and Walukiewicz (2015, Section 5) give another construction proving Lemma 10.3. It has the disadvantage that the resulting scheme $\mathcal{G}_{\mathcal{M}}$ is not of the same order as $\mathcal{G}$; the order grows by one. For this reason we prefer to refer to the construction of Haddad.

Word-recognizing schemes. In the remaining part of this section we show how Theorem 10.2 follows from Lemma 10.3. We first solve the case of word-recognizing schemes. To this end, fix some number $m \in \mathbb{N}$ and some word-recognizing scheme $\mathcal{G}$ of order at most $m+1$ (notice the shift by 1 with respect to the statement of the theorem).

Having in mind some fixed m, we define the *value* of a closed lambda-term P, denoted $[\![P]\!]_{\mathcal{M}_m}$, as the set of pairs $(A, \hat{\tau})$ for which there exists an A-pumpable derivation of $\varepsilon \vdash_m P : \hat{\tau} \triangleright c$ for some c. When P is of sort α, $[\![P]\!]_{\mathcal{M}_m}$ belongs to the finite set $D_m[\alpha] = \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{TT}^{\alpha}_m)$.

We remark that $[\![P]\!]_{\mathcal{M}_m}$ contains in particular pairs of the form $(\emptyset, \hat{\tau})$ (i.e., with $A = \emptyset$), which simply contain all type triples that can be derived for P.

We now equip D_m with operations of application and abstraction, and we prove (in Lemmata 10.4-10.5) that $\mathcal{M}_m$ is a morphism. For every sort of the form $\alpha \to \beta$, and for $\chi \in D_m[\alpha \to \beta]$ and $\chi' \in D_m[\alpha]$, let $\chi\,\chi' \in D_m[\beta]$ be the set of pairs $(A, \hat{\tau})$ such that for some pairs $(A_0, \hat{\sigma}_0) \in \chi$ and $(A_1, \hat{\sigma}_1), \dots, (A_n, \hat{\sigma}_n) \in \chi'$ with $A = A_0 \uplus A_1 \uplus \dots \uplus A_n$ one can apply the (@) rule to type judgments $\varepsilon \vdash_m P : \hat{\sigma}_0 \triangleright c_0$ and $\varepsilon \vdash_m Q : \hat{\sigma}_i \triangleright c_i$ for $i \in \{1, \dots, n\}$ and derive $\varepsilon \vdash_m P\,Q : \hat{\tau} \triangleright c$ (for some lambda-terms $P^{\alpha \to \beta}, Q^{\beta}$, and for some flag counters $c, c_0, c_1, \dots, c_n$; they are irrelevant here). Additionally, for every sort of the form $\alpha \to \beta$, and for $\chi \in D_m[\beta]$, let $\boldsymbol{\lambda}\alpha.\chi \in D_m[\alpha \to \beta]$ be the set of pairs $(A, \hat{\tau})$ such that for some $(A, \hat{\tau}') \in \chi$ one can apply the (λ) rule to $\varepsilon \vdash_m P : \hat{\tau}' \triangleright c$ and derive $\varepsilon \vdash_m \lambda x.P : \hat{\tau} \triangleright c$ (for some lambda-term $\lambda x^{\alpha}.P^{\beta}$ with P being closed, and for some flag counter c; they are irrelevant here).

Lemma 10.4. *For every closed lambda-term of the form $P\,Q$ it is the case that $[\![P]\!]_{\mathcal{M}_m}\,[\![Q]\!]_{\mathcal{M}_m} = [\![P\,Q]\!]_{\mathcal{M}_m}$.*

Proof: Suppose that $(A, \hat{\tau}) \in [\![P\,Q]\!]_{\mathcal{M}_m}$. This implies that there exists an A-pumpable derivation tree t that derives $\varepsilon \vdash_m P\,Q : \hat{\tau} \triangleright c$ for some c. By the definition of A-pumpability, for every $a \in A$ there is a pair of nodes u_a, v_a of t such that u_a is an ancestor of v_a, the type judgments in u_a and in v_a are equivalent, and the flag counters in u_a and in v_a differ on the a coordinate.

First, let us justify that without loss of generality we can assume that none of u_a points to the root of t. To this end, suppose that u_a for some a is located in the root of t. Then we copy the fragment of t lying between u_a and v_a, we adjust appropriately flag counters on the path from u_a to v_a in the copy, and we attach the copy above u_a (i.e., above the previous root). In this way we obtain a new derivation tree t', which derives $\varepsilon \vdash_m P\,Q : \hat{\tau} \triangleright c'$ for some c', and which is again A-pumpable, but now none of u_a equals the root of t', because all u_a and v_a belong to t, which is a proper subtree of t'.

Thus, we assume that none of u_a points to the root of t. Let $w_0, w_1, \dots, w_n$ be the children of the root of t (with w_0 corresponding to P), and, for $i \in \{0, \dots, n\}$, let A_i be the set of those $a \in A$ for which u_a and v_a belong to the subtree of t starting in w_i. Clearly $A = A_0 \uplus A_1 \uplus \dots \uplus A_n$, and the subtree starting in w_i is A_i-pumpable, for every $i \in \{0, \dots, n\}$. If the type judgment in w_i is $\varepsilon \vdash_m R_i : \hat{\sigma}_i \triangleright c_i$ (where R_i

equals either P or Q), we have that $(A_i, \hat{\sigma}_i) \in [\![R_i]\!]_{\mathcal{M}_m}$. By the definition of the application operation in D_m, it follows that $(A, \hat{\tau}) \in [\![P]\!]_{\mathcal{M}_m}\, [\![Q]\!]_{\mathcal{M}_m}$.

For the opposite inclusion, consider a pair $(A, \hat{\tau}) \in [\![P]\!]_{\mathcal{M}_m}\, [\![Q]\!]_{\mathcal{M}_m}$. By the definition of the application operation in D_m, there are some pairs $(A_0, \hat{\sigma}_0) \in [\![P]\!]_{\mathcal{M}_m}$ and $(A_1, \hat{\sigma}_1), \ldots, (A_n, \hat{\sigma}_n) \in [\![Q]\!]_{\mathcal{M}_m}$ with $A = A_0 \uplus A_1 \uplus \cdots \uplus A_n$ such that one can apply the (@) rule to type judgments $\varepsilon \vdash_m P' : \hat{\sigma}_0 \triangleright c'_0$ and $\varepsilon \vdash_m Q' : \hat{\sigma}_i \triangleright c'_i$ for $i \in \{1, \ldots, n\}$ and derive $\varepsilon \vdash_m P\,Q : \hat{\tau} \triangleright c'$, for some lambda-terms P', Q' and some flag counters $c', c'_0, c'_1, \ldots, c'_n$. Moreover, by the definition of $[\![P]\!]_{\mathcal{M}_m}$ and $[\![Q]\!]_{\mathcal{M}_m}$, there exists an A_0-pumpable derivation of $\varepsilon \vdash_m P : \hat{\sigma}_0 \triangleright c_0$, and A_i-pumpable derivations of $\varepsilon \vdash_m Q : \hat{\sigma}_i \triangleright c_i$ for $i \in \{1, \ldots, n\}$ for some flag counters $c_0, c_1, \ldots, c_n$. Notice that in the (@) rule we can be harmlessly change the lambda-terms P', Q' to P, Q, and the flag counters $c'_0, c'_1, \ldots, c'_n$ to $c_0, c_1, \ldots, c_n$. Thus, by attaching the (@) rule to our A_i-pumpable derivations, we obtain an A-pumpable derivation of $\varepsilon \vdash_m P\,Q : \hat{\tau} \triangleright c$, for some flag counter c. This implies that $(A, \hat{\tau}) \in [\![P\,Q]\!]_{\mathcal{M}_m}$. □

Lemma 10.5. *For every lambda-term of the form $\lambda x^{\alpha}.P$, where P is closed, it is the case that $[\![\lambda x.P]\!]_{\mathcal{M}_m} = \boldsymbol{\lambda}\alpha.[\![P]\!]_{\mathcal{M}_m}$.*

Proof: Suppose that $(A, \hat{\tau}) \in [\![\lambda x.P]\!]_{\mathcal{M}_m}$. This implies that there exists an A-pumpable derivation tree t that derives $\varepsilon \vdash_m \lambda x.P : \hat{\tau} \triangleright c$ for some c. By the definition of A-pumpability, for every $a \in A$ there is a pair of nodes u_a, v_a of t such that u_a is an ancestor of v_a, the type judgments in u_a and in v_a are equivalent, and the flag counters in u_a and in v_a differ on the a coordinate. As in the proof of Lemma 10.4, we can assume that none of u_a points to the root of t. Then the subtree of t starting in the only child of the root is an A-pumpable derivation tree that derives $\varepsilon \vdash_m P : \hat{\tau}' \triangleright c$ for some $\hat{\tau}'$, that is, $(A, \hat{\tau}') \in [\![P]\!]_{\mathcal{M}_m}$. Moreover, $\varepsilon \vdash_m \lambda x.P : \hat{\tau} \triangleright c$ is derived by applying the (λ) rule to $\varepsilon \vdash_m P : \hat{\tau}' \triangleright c$. By the definition of the abstraction operation it follows that $(A, \hat{\tau}) \in \boldsymbol{\lambda}\alpha.[\![P]\!]_{\mathcal{M}_m}$.

Conversely, suppose that $(A, \hat{\tau}) \in \boldsymbol{\lambda}\alpha.[\![P]\!]_{\mathcal{M}_m}$. By definition this means that, for some lambda-term P' of the same sort as P, and for some flag counter c', one can derive $\varepsilon \vdash_m \lambda x.P' : \hat{\tau} \triangleright c'$ by applying the (λ) rule to $\varepsilon \vdash_m P' : \hat{\tau}' \triangleright c'$ for some $\hat{\tau}'$ such that $(A, \hat{\tau}') \in [\![P]\!]_{\mathcal{M}_m}$. Moreover, by the definition of $[\![P]\!]_{\mathcal{M}_m}$, there exists an A-pumpable derivation of $\varepsilon \vdash_m P : \hat{\tau}' \triangleright c$ for some flag counter c. Notice that in the (λ) rule we can harmlessly change the lambda-term P' to P, and the flag counter c' to c. Thus, by attaching the (λ) rule to our derivation, we obtain an A-pumpable derivation of $\varepsilon \vdash_m \lambda x.P : \hat{\tau} \triangleright c$, witnessing that $(A, \hat{\tau}) \in [\![\lambda x.P]\!]_{\mathcal{M}_m}$. □

Lemmata 10.4-10.5 show that $\mathcal{M}_m$ is a morphism, so we are now ready to apply Lemma 10.3 to the considered word-recognizing scheme $\mathcal{G}$ of order at most $m+1$. The lemma gives us a scheme $\mathcal{G}_{\mathcal{M}_m}$ of order at most $m+1$ such that $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m}))$ is a $\Xi^{m+1}_{\mathcal{M}_m}$-reflection of $BT(\Lambda(\mathcal{G}))$.

In order to obtain a Ξ_{SUP}-reflection, it remains to relabel nodes of the $\Xi^{m+1}_{\mathcal{M}_m}$-reflection. Namely, we change the label in every node constructor of $\mathcal{G}_{\mathcal{M}_m}$ from (a, χ) to $(a, \{A \mid (A, \hat{\rho}^{all}_m) \in \chi\})$; there is one exception: node constructors of the form $(\mathsf{nd}, \chi)\langle\rangle$ are changed to $\mathsf{nd}\langle\rangle$. The resulting scheme, called $\mathcal{G}_{SUP}$, is of order at most $m+1$, as needed.

Correctness of the construction is a consequence of the following simple lemma.

Lemma 10.6. *If P is a word-recognizing closed lambda-term of sort o and of complexity at most $m+1$ then, for every set $A \subseteq \Sigma$, it is the case that $(A, \hat{\rho}^{all}_m) \in [\![P]\!]_{\mathcal{M}_m}$ if and only if $SUP_A(\mathcal{L}(BT(P)))$.*

Proof: By definition, a pair $(A, \hat{\rho}^{all}_m)$ belongs to $[\![P]\!]_{\mathcal{M}_m}$ if and only if there is an A-pumpable derivation of $\varepsilon \vdash_m P : \hat{\rho}^{all}_m \triangleright c$ for some c. By Lemma 9.9 the latter holds if and only if for every $n \in \mathbb{N}$ we can

derive $\varepsilon \vdash_m P : \hat{\rho}_m^{all} \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$. This in turn, by Theorem 3.2, is equivalent to $SUP_A(\mathcal{L}(BT(P)))$, that is, to $SUP_A(\mathcal{L}(T))$. We remark that in order to use Theorem 3.2 we indeed need to know that P is word-recognizing and of complexity at most $m+1$. □

Using the above lemma, we now justify that $BT(\Lambda(\mathcal{G}_{SUP}))$ is the Ξ_{SUP}-reflection of $BT(\Lambda(\mathcal{G}))$ (i.e., we prove correctness of the construction). Clearly $BT(\Lambda(\mathcal{G}_{SUP}))$, $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m}))$, and $BT(\Lambda(\mathcal{G}))$ are of the same shape. Consider now a subtree T of $BT(\Lambda(\mathcal{G}))$, and the roots $r_{\mathcal{M}_m}$ and r_{SUP} of the corresponding subtrees of $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m}))$ and $BT(\Lambda(\mathcal{G}_{SUP}))$, respectively. If $T = \mathsf{nd}\langle\rangle$, then $r_{\mathcal{M}_m}$ is either labeled by nd or by (nd, χ) for some χ, and thus r_{SUP} is labeled by nd. Otherwise, the label of $r_{\mathcal{M}_m}$ is $(a, [\![P]\!]_{\mathcal{M}_m})$, where a is the label of the root of T, and P is some closed lambda-term of sort o and of complexity at most $m+1$ such that $BT(P) = T$. In consequence, the label of r_{SUP} is $(a, \{A \mid (A, \hat{\rho}_m^{all}) \in [\![P]\!]_{\mathcal{M}_m}\})$. Observe that P is word-recognizing: $BT(P) = T$ is a subtree of $BT(\Lambda(\mathcal{G}))$, so if all elements of $\mathcal{L}(BT(\Lambda(\mathcal{G})))$ are words, also all elements of $\mathcal{L}(BT(P))$ are words. By Lemma 10.6, $(A, \hat{\rho}_m^{all}) \in [\![P]\!]_{\mathcal{M}_m}$ is equivalent to $SUP_A(\mathcal{L}(BT(P)))$, that is, to $SUP_A(\mathcal{L}(T))$. This implies that $BT(\Lambda(\mathcal{G}_{SUP}))$ is indeed the Ξ_{SUP}-reflection of $BT(\Lambda(\mathcal{G}))$.

It remains to bound the running time. In this part, we assume that $m \geq 1$. Considering $m = 0$ does not make sense, as anyway we want to prove that the complexity is at most $\max(m,1)$-fold exponential in $|\mathcal{G}|$ (and at most doubly exponential in $|\Sigma|$). Moreover, without loss of generality we assume that $|\Sigma| \leq |\mathcal{G}|$; letters not appearing in $\mathcal{G}$ are irrelevant anyway. First, we state a general lemma.

Lemma 10.7. *Let $m \geq 1$. Suppose that we are given a scheme $\mathcal{G}$ of order at most $m+1$, and that $|\Sigma| \leq |\mathcal{G}|$.*

1. *For a sort $\alpha \in \mathcal{S}_{\mathcal{G}}$ such that $ord(\alpha) \leq m$, the size of $D_m[\alpha]$ is at most m-fold exponential in $|\mathcal{G}|$ and doubly exponential in $|\Sigma|$.*
2. *Given a closed lambda-term P of complexity at most $m+1$ and of size polynomial in $|\mathcal{G}|$, one can compute $[\![\Lambda_{\mathcal{G}}(P)]\!]_{\mathcal{M}_m}$ in time m-fold exponential in $|\mathcal{G}|$.*
3. *Given two elements $\chi \in D[\alpha \to \beta], \chi' \in D[\alpha]$, where $\alpha \to \beta$ is of order at most $m+1$ and of size polynomial in $|\mathcal{G}|$, one can compute $\chi\,\chi'$ in time m-fold exponential in $|\mathcal{G}|$.*
4. *Given an element $\chi \in D[\beta]$ and a sort α, where $\alpha \to \beta$ is of order at most $m+1$ and of size polynomial in $|\mathcal{G}|$, one can compute $\boldsymbol{\lambda}\alpha.\chi$ in time m-fold exponential in $|\mathcal{G}|$.*

Proof: Take a sort $\alpha \in \mathcal{S}_{\mathcal{G}}$ of order at most m. Recalling that $D_m[\alpha] = \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{TT}_m^{\alpha})$ we see that $|D_m[\alpha]| = 2^{2^{|\Sigma|} \cdot |\mathcal{TT}_m^{\alpha}|}$. In Section 9.1 we have defined a number $\eta_m^{\mathcal{G}}$ such that $|\mathcal{TT}_m^{\alpha}| \leq \eta_m^{\mathcal{G}}$ (assuming $\alpha \in \mathcal{S}_{\mathcal{G}}$ and $ord(\alpha) \leq m$), and we have shown in Lemma 9.6 that $\eta_m^{\mathcal{G}}$ is at most $(m-1)$-fold exponential in $|\mathcal{G}|$ (where "0-fold exponential" means "polynomial"). In consequence, $D_m[\alpha]$ is at most m-fold exponential in $|\mathcal{G}|$ and doubly exponential in $|\Sigma|$, establishing Point 1.

Concerning Point 2, recall that $[\![\Lambda_{\mathcal{G}}(P)]\!]_{\mathcal{M}_m}$ is the set set of pairs $(A, \hat{\tau})$ for which there exists an A-pumpable derivation of $\varepsilon \vdash_m \Lambda_{\mathcal{G}}(P) : \hat{\tau} \triangleright c$ for some c. We check, for every pair separately, whether this is the case using the algorithm given by Lemma 9.13. Formally, the algorithm works only for subterms of $\Lambda(\mathcal{G})$. We thus consider a modified scheme $\mathcal{G}'$, which is obtained from $\mathcal{G}$ by adding a fresh starting nonterminal N_P with rule $\mathcal{R}(N_P) = P\,R_1\,\ldots\,R_s$ for some lambda-terms $R_1, \ldots, R_s$ of sorts $\alpha_1, \ldots, \alpha_s$, respectively, where $\alpha_1 \to \ldots \to \alpha_s \to o$ is the sort of P (if P is a nonterminal of sort o, instead of that we set P to be the starting nonterminal, in order to avoid a forbidden situation when $\mathcal{R}(N_P)$ is a nonterminal); then $\Lambda_{\mathcal{G}}(P)$ is a subterm of $\Lambda(\mathcal{G}')$, and $|\mathcal{G}'|$ is polynomial in $|\mathcal{G}|$. For every pair the running time is at most m-fold exponential in $|\mathcal{G}|$. The number of sets A to consider is at most $2^{|\Sigma|}$, and the number of type triples

$\hat{\tau}$ to consider is bounded by $\eta^{\mathcal{G}'}_{m+1}$ (they belong to $\mathcal{TT}^{\alpha}_{m}$, where $\alpha \in \mathcal{S}_{\mathcal{G}'}$ is the sort of $\Lambda_{\mathcal{G}}(P)$, being of order at most $m+1$), so it is at most m-fold exponential in $|\mathcal{G}'|$ (i.e., in $|\mathcal{G}|$) by Lemma 9.6.

In Point 3 we should compute an application $\chi\,\chi'$ given χ and χ'. Again, by considering a modified scheme $\mathcal{G}'$ we may assume that $\Lambda(\mathcal{G}')$ has subterms of sort $\alpha \to \beta$ and α. Suppose that we want to check whether $(A, \hat{\tau}) \in (\chi\,\chi')$. To this end, we need to choose pairs $(A_0, \hat{\sigma}_0) \in \chi$ and $(A_1, \hat{\sigma}_1), \ldots, (A_k, \hat{\sigma}_k) \in \chi'$ such that A_i for $i \in \{1, \ldots, k\}$ are nonempty and $A = A_0 \uplus A_1 \uplus \cdots \uplus A_k$. As in Point 2, for every $i \in \{0, \ldots, k\}$ the number of pairs $(A_i, \hat{\sigma}_i)$ to consider is at most m-fold exponential in $|\mathcal{G}|$; moreover $k \leq |A| \leq |\Sigma| \leq |\mathcal{G}|$, so altogether the number of choices is also at most m-fold exponential in $|\mathcal{G}|$. Then, we need to check whether the type triple $\hat{\tau}$ can be derived by the (@) rule when for the operator we use the type triple $\hat{\sigma}_0$, and for the argument we use the type triples $\hat{\sigma}_1, \ldots, \hat{\sigma}_k$ and, additionally, an arbitrary number of type triples $\hat{\sigma}$ such that $(\emptyset, \hat{\sigma}) \in \chi'$. Such a check can be performed using Lemma 9.12.

Point 4 is even a bit simpler. Again, we imagine a scheme $\mathcal{G}'$ such that $\Lambda(\mathcal{G}')$ has a subterm $\lambda x.P$ of sort $\alpha \to \beta$. Then, for every pair $(A, \hat{\tau}') \in \chi$ and for every $\hat{\tau} \in \mathcal{T}^{\alpha \to \beta}_{m}$ we have to check whether by applying the (λ) rule to $\varepsilon \vdash_m P : \hat{\tau}' \triangleright \mathbf{0}$ one can derive the type triple $\hat{\tau}$ for $\lambda x.P$; this can be done using Lemma 9.12. □

According to Lemma 10.3, the time needed to construct $\mathcal{G}_{\mathcal{M}_m}$ is polynomial in five ingredients. The first of them is $|\mathcal{G}_{\mathcal{M}_m}|$. Recalling that we have shifted the order m by one, from Lemma 10.3 it follows that

$$|\mathcal{G}_{\mathcal{M}_m}| \leq |\mathcal{G}| \cdot \left(\max_{\substack{\alpha \in \mathcal{S}_{\mathcal{G}} \\ ord(\alpha) \leq m}} |D_m[\alpha]| \right)^{|\mathcal{G}|^2}.$$

By Point 1 of Lemma 10.7, all $|D_m[\alpha]|$ appearing in the above formula are at most m-fold exponential in $|\mathcal{G}|$ and doubly exponential in $|\Sigma|$, thus the same is true for $|\mathcal{G}_{\mathcal{M}_m}|$. Beside of that, we should bound the running time described by Points (A)-(D) of Lemma 10.3. In Lemma 10.7 we have already said that the time needed to compute $[\![\Lambda_{\mathcal{G}}(P)]\!]_{\mathcal{M}_m}$, and $\chi\,\chi'$, and $\boldsymbol{\lambda}\alpha.\chi$ is m-fold exponential in $|\mathcal{G}|$ (as needed for Points (A), (B), and (C), respectively). Moreover, elements of $D_m[\alpha]$ can be enumerated in time polynomial in $|D_m[\alpha]|$, in a trivial way; according to Point (D) this is needed only when $\alpha \in \mathcal{S}_{\mathcal{G}}$ and $ord(\alpha) \leq m$, so $|D_m[\alpha]|$ is already included in the inequality for $|\mathcal{G}_{\mathcal{M}_m}|$. Altogether we obtain that $\mathcal{G}_{\mathcal{M}_m}$ can be constructed in time at most m-fold exponential in $|\mathcal{G}|$ and doubly exponential in $|\Sigma|$. The same bound applies to $\mathcal{G}_{SUP}$, because while converting $\mathcal{G}_{\mathcal{M}_m}$ into $\mathcal{G}_{SUP}$ we only change some labels in a straightforward way. This finishes the proof of Theorem 10.2 for word-recognizing schemes.

Tree-recognizing schemes. Next, we prove Theorem 10.2 for tree-recognizing schemes. To this end, fix some number $m \in \mathbb{N}$, and some scheme $\mathcal{G}$ of order at most m. We are going to use the translation from Section 8. We remark, though, that it is not enough to convert $\mathcal{G}$ into a word-recognizing scheme $\mathcal{G}^\flat$, and apply the word variant of Theorem 10.2 to $\mathcal{G}^\flat$: this would only give a reflection of $\mathcal{G}^\flat$, while we want a reflection of $\mathcal{G}$. Thus, we rather modify the original scheme $\mathcal{G}$, and we only apply the translation while calculating values.

Namely, we define the $\flat$-*value* of a closed lambda-term P, denoted $[\![P]\!]_{\mathcal{M}^\flat_m}$, as $[\![P^\flat]\!]_{\mathcal{M}_m}$. When P is of sort α, $[\![P]\!]_{\mathcal{M}^\flat_m}$ belongs to the set $D^\flat_m[\alpha] = D_m[\alpha^\flat]$. The application operation in $D^\flat_m$ is inherited from $\mathcal{D}_m$, and the abstraction operation $\boldsymbol{\lambda}^\flat$ in $D^\flat_m$ is defined by $\boldsymbol{\lambda}^\flat\alpha.\chi = \boldsymbol{\lambda}\alpha^\flat.\chi$. It easily follows from Lemmata 10.4 and 10.5 that $\mathcal{M}^\flat_m$ is a morphism. Indeed, on the one hand, for every closed lambda-term of

the form $P\,Q$ it is the case that

$$[\![P]\!]_{\mathcal{M}_m^\flat}\,[\![Q]\!]_{\mathcal{M}_m^\flat} = [\![P^\flat]\!]_{\mathcal{M}_m}\,[\![Q^\flat]\!]_{\mathcal{M}_m} = [\![P^\flat\,Q^\flat]\!]_{\mathcal{M}_m} = [\![(P\,Q)^\flat]\!]_{\mathcal{M}_m} = [\![P\,Q]\!]_{\mathcal{M}_m^\flat}\,.$$

On the other hand, for every lambda-term of the form $\lambda x^\alpha.P$, where P is closed, it is the case that

$$[\![\lambda x.P]\!]_{\mathcal{M}_m^\flat} = [\![(\lambda x.P)^\flat]\!]_{\mathcal{M}_m} = [\![\lambda x^\flat.P^\flat]\!]_{\mathcal{M}_m} = \boldsymbol{\lambda}\alpha^\flat.[\![P^\flat]\!]_{\mathcal{M}_m} = \boldsymbol{\lambda}^\flat\alpha.[\![P]\!]_{\mathcal{M}_m^\flat}\,.$$

Lemma 10.3 applied to $\mathcal{G}$ and to the morphism $\mathcal{M}_m^\flat$ gives us a scheme $\mathcal{G}_{\mathcal{M}_m^\flat}$ of order at most m such that $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m^\flat}))$ is a $\Xi^m_{\mathcal{M}_m^\flat}$-reflection of $BT(\Lambda(\mathcal{G}))$.

Then, we change the label in every node constructor of $\mathcal{G}_{\mathcal{M}_m^\flat}$ from (a,χ) to $(a,\{A \mid (A,\hat\rho_m^{all}) \in (\chi\,[\![\mathsf{e}\langle\rangle]\!]_{\mathcal{M}_m})\})$, with the exception that we change node constructors of the form $(\mathsf{nd},\chi)\langle\rangle$ to $\mathsf{nd}\langle\rangle$. The resulting scheme, called $\mathcal{G}_{SUP}$, is of order at most m, as needed.

Let us now prove that $BT(\Lambda(\mathcal{G}_{SUP}))$ is indeed the Ξ_{SUP}-reflection of $BT(\Lambda(\mathcal{G}))$. By construction, $BT(\Lambda(\mathcal{G}_{SUP}))$, $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m^\flat}))$, and $BT(\Lambda(\mathcal{G}))$ are of the same shape. Consider a subtree T of $BT(\Lambda(\mathcal{G}))$, and the roots $r_{\mathcal{M}_m^\flat}$ and r_{SUP} of the corresponding subtrees of $BT(\Lambda(\mathcal{G}_{\mathcal{M}_m^\flat}))$ and $BT(\Lambda(\mathcal{G}_{SUP}))$, respectively. If $T = \mathsf{nd}\langle\rangle$, then $r_{\mathcal{M}_m^\flat}$ is either labeled by nd or by (nd,χ) for some χ, and thus r_{SUP} is labeled by nd. Otherwise, the label of $r_{\mathcal{M}_m^\flat}$ is $(a,[\![P]\!]_{\mathcal{M}_m^\flat})$, where a is the label of the root of T, and P is some closed lambda-term of sort o and of complexity at most m such that $BT(P) = T$. In consequence, the label of r_{SUP} is $(a,\{A \mid (A,\hat\rho_m^{all}) \in ([\![P]\!]_{\mathcal{M}_m^\flat}\,[\![\mathsf{e}\langle\rangle]\!]_{\mathcal{M}_m})\})$. By Lemma 10.4, $[\![P]\!]_{\mathcal{M}_m^\flat}\,[\![\mathsf{e}\langle\rangle]\!]_{\mathcal{M}_m} = [\![P^\flat]\!]_{\mathcal{M}_m}\,[\![\mathsf{e}\langle\rangle]\!]_{\mathcal{M}_m} = [\![P^\flat\,(\mathsf{e}\langle\rangle)]\!]_{\mathcal{M}_m}$. By construction $P^\flat\,\mathsf{e}\langle\rangle$ is word-recognizing, and of complexity at most $m+1$. Thus, by Lemma 10.6, $(A,\hat\rho_m^{all}) \in [\![P^\flat\,(\mathsf{e}\langle\rangle)]\!]_{\mathcal{M}_m}$ is equivalent to $SUP_A(\mathcal{L}(BT(P^\flat\,(\mathsf{e}\langle\rangle))))$. By Point 1(b) of Lemma 8.1, the latter is equivalent to $SUP_A(\mathcal{L}(BT(P)))$, that is, to $SUP_A(\mathcal{L}(T))$, which finishes the proof.

Finally, we bound the running time needed to construct $\mathcal{G}_{\mathcal{M}_m^\flat}$, and thus $\mathcal{G}_{SUP}$. As for words, we assume here that $m \geq 1$; this is without loss of generality, because anyway we want to prove that the complexity is at most $\max(m,1)$-fold exponential in $|\mathcal{G}|$ (and at most doubly exponential in $|\Sigma|$). Point 1 of Lemma 10.3 implies that

$$|\mathcal{G}_{\mathcal{M}_m^\flat}| \leq |\mathcal{G}| \cdot \left(\max_{\substack{\alpha\in\mathcal{S}_\mathcal{G}\\ ord(\alpha)\leq m-1}} \left| D_m^\flat[\alpha] \right| \right)^{|\mathcal{G}|^2}\,.$$

We recall that $D_m^\flat[\alpha] = D_m[\alpha^\flat]$, that $ord(\alpha^\flat) = ord(\alpha)+1$ (so $ord(\alpha) \leq m-1$ implies $ord(\alpha^\flat) \leq m$), and that $\alpha \in \mathcal{S}_\mathcal{G}$ implies $\alpha^\flat \in \mathcal{S}_{\mathcal{G}^\flat}$. In consequence, Point 1 of Lemma 10.7 applied to the scheme $\mathcal{G}^\flat$ (which is of order at most $m+1$, and of size linear in $|\mathcal{G}|$), implies that all $|D_m^\flat[\alpha]| = |D_m[\alpha^\flat]|$ appearing in the above formula are at most m-fold exponential in $|\mathcal{G}|$ and doubly exponential in $|\Sigma|$. The same is true for $|\mathcal{G}_{\mathcal{M}_m^\flat}|$. The other ingredients of the running time can be bounded as follows.

(A) We need to compute the value $[\![\Lambda_\mathcal{G}(P)]\!]_{\mathcal{M}_m^\flat}$, given a closed lambda-term P of complexity at most m and of size polynomial in $|\mathcal{G}|$. Such a value equals $[\![\Lambda_{\mathcal{G}^\flat}(P^\flat)]\!]_{\mathcal{M}_m}$, where $P^\flat$ is of complexity at most $m+1$. Because $\mathcal{G}^\flat$ is of order at most $m+1$, the latter value can be computed in time m-fold exponential in $|\mathcal{G}^\flat|$ by Lemma 10.7, Point 2.

(B) We need to compute the composition $\chi\,\chi'$, given two elements $\chi \in D_m^\flat[\alpha\to\beta], \chi' \in D_m^\flat[\alpha]$, where $\alpha\to\beta$ is of order at most m and of size polynomial in $|\mathcal{G}|$. Recall that the composition in $\mathcal{M}_m^\flat$ is

inherited from $\mathcal{M}_m$. We notice that $D^\flat_m[\alpha \to \beta] = D_m[\alpha^\flat \to \beta^\flat]$, and $D^\flat_m[\alpha] = D_m[\alpha^\flat]$, and that $\alpha^\flat \to \beta^\flat$ is of order at most $m+1$, so the composition can be computed in time m-fold exponential in $|\mathcal{G}^\flat|$ by Lemma 10.7, Point 3.

(C) Given an element $\chi \in D^\flat_m[\beta] = D_m[\beta^\flat]$ and a sort α, where $\alpha \to \beta$ is of order at most m and of size polynomial in $|\mathcal{G}|$, we need to compute the abstraction $\boldsymbol{\lambda}^\flat.\chi = \boldsymbol{\lambda}\alpha^\flat.\chi$. This can be done in time m-fold exponential in $|\mathcal{G}|$ by Lemma 10.7, Point 4 (the order of $\alpha^\flat \to \beta^\flat$ is at most $m+1$).

(D) We need to enumerate all elements of $D_{\mathcal{M}^\flat_m}[\alpha]$ for $\alpha \in \mathcal{S}_\mathcal{G}$ such that $ord(\alpha) \leq m-1$. This is trivial (recalling that $D_{\mathcal{M}^\flat_m}[\alpha] = D_{\mathcal{M}_m}[\alpha^\flat]$) and the size of $D_{\mathcal{M}^\flat_m}[\alpha]$ is already included in the inequality for $|\mathcal{G}_{\mathcal{M}^\flat_m}|$.

Altogether, from Lemma 10.3 it follows that the running time is at most m-fold exponential in $|\mathcal{G}^\flat|$ and doubly exponential in $|\Sigma|$, as needed for Theorem 10.2.

11 Downward closure

The downward closure of a language of words L, denoted $L{\downarrow}$, is the set of all scattered subwords (subsequences) of words from L. Recall that the downward closure of any set is always a regular language; moreover, it is a finite union of *ideals*, that is, languages of the form $Y_0^*\{x_1,\varepsilon\}Y_1^*\ldots\{x_n,\varepsilon\}Y_n^*$, where $x_1,\ldots,x_n$ are letters, and $Y_0,\ldots,Y_n$ are sets of letters. The main interest on SUP comes from the fact that this problem is closely related to computability of the downward closure of languages of words (where we aim in presenting the result by a list of ideals, or by a finite automaton). Indeed, having a word-recognizing scheme $\mathcal{G}$, it is not difficult to compute $\mathcal{L}(\mathcal{G}){\downarrow}$ by performing multiple calls to a procedure solving SUP (for products of $\mathcal{G}$ and some finite automata). The complexity of this algorithm is directly related to the size of its output. We, however, do not know any upper bound on the size of (a representation of) $\mathcal{L}(\mathcal{G}){\downarrow}$. A recently developed pumping lemma for nondeterministic schemes (Asada and Kobayashi, 2017) may shed some new light on this subject (while pumping lemmata for deterministic schemes (Kartzow and Parys, 2012; Kobayashi, 2013) seem irrelevant here).

Instead of actually computing the downward closure, Zetzsche (2016) proposed to consider the following decision problem of downward-closure inclusion: given two word-recognizing schemes $\mathcal{G},\mathcal{H}$ of order at most m, check whether $\mathcal{L}(\mathcal{G}){\downarrow} \subseteq \mathcal{L}(\mathcal{H}){\downarrow}$; he proved that this problem is co-m-NEXPTIME-hard. It would be interesting to give some upper bound on the complexity of this problem. Although, again, we do not know how to do this, we can at least give a partial result.

Theorem 11.1. *Let $m \geq 2$. Given a word-recognizing scheme $\mathcal{H}$ of order at most m, and an ideal I, the problem of deciding whether $I \subseteq \mathcal{L}(\mathcal{H}){\downarrow}$ is $(m-1)$-EXPTIME-complete.*

We remark that schemes of order 0 are equivalent to nondeterministic finite automata, and schemes of order at most 1 are equivalent to context-free grammars (and translations between these formalisms can be performed in logarithmic space). Thus from Zetzsche (2016) it follows that the problem of deciding whether $I \subseteq \mathcal{L}(\mathcal{H}){\downarrow}$ is NL-complete for $\mathcal{H}$ of order 0, and P-complete for $\mathcal{H}$ of order at most 1.

Proof of Theorem 11.1: Let us first see that the problem is $(m-1)$-EXPTIME-hard. This follows directly from $(m-1)$-EXPTIME-hardness of the problem of deciding whether $\mathcal{L}(\mathcal{H})$ is nonempty (cf. Lemma 9.14). Indeed, $\mathcal{L}(\mathcal{H}) \neq \emptyset$ if and only if $\{\varepsilon\} \subseteq \mathcal{L}(\mathcal{H}){\downarrow}$; we notice that the singleton containing the empty word is a special case of an ideal.

In the remaining part of this section we prove that the problem can be actually solved in $(m-1)$-EXPTIME. We follow here the approach of Zetzsche (2015, 2016). He has shown (Zetzsche, 2015, Proof

of Theorem 1) that based on an ideal I one can construct a nondeterministic finite-state transducer $\mathcal{T}$ and a set of letters A such that for every language L we have that $I \subseteq L{\downarrow}$ if and only if $SUP_A(\mathcal{T}(L))$ holds.[(vii)] Here by $\mathcal{T}(L)$ we mean the effect of applying the transformation defined by $\mathcal{T}$ to the language L (i.e., the set of all words w such that for some $v \in L$ the pair (v, w) is in the relation recognized by $\mathcal{T}$). The construction of $\mathcal{T}$ and A can be performed in polynomial time, and SUP for word-recognizing schemes of order at most m can be solved in $(m-1)$-EXPTIME.

It remains to see that $\mathcal{H}$ and $\mathcal{T}$ can be combined (in polynomial time) into a scheme $\mathcal{H}_\mathcal{T}$ such that $\mathcal{L}(\mathcal{H}_\mathcal{T}) = \mathcal{T}(\mathcal{L}(\mathcal{H}))$. To this end, we perform the following steps.

- Treating nd as any other letter, we translate $\mathcal{H}$ into a deterministic tree-generating collapsible pushdown automaton (CPDA) $\mathcal{A}$ that generates $BT(\Lambda(\mathcal{H}))$. Preferably, we refer here to the translation of Salvati and Walukiewicz (2016, Sections 3.1 and 4), as this translation is given for schemes defined similarly as in the current paper, and thus it can be easily adapted. In particular, it works well when as $\mathcal{R}(N)$ we allow arbitrary lambda-terms (cf. Appendix A.3). It can be seen that their translation works in polynomial time. We shall only remark that the size of λY-terms (appearing in their paper as intermediate objects) should be defined as the number of different subterms; in other words, λY-terms should be represented as (directed, acyclic) graphs, without expanding them into trees.
- We change the deterministic tree-generating CPDA $\mathcal{A}$ into a nondeterministic word-recognizing CPDA $\mathcal{B}$: whenever $\mathcal{A}$ was generating a node with nd as its label and with r children, in $\mathcal{B}$ we nondeterministically choose one of the r options; whenever $\mathcal{A}$ was generating a node with some other letter as its label (and with at most one child), in $\mathcal{B}$ we allow to read this letter, and if this node had no children, we accept. As a result of this construction we obtain an automaton $\mathcal{B}$ that recognizes the language $\mathcal{L}(\mathcal{H})$, seen as a language of words.
- We combine $\mathcal{B}$ with our finite-state transducer $\mathcal{T}$, so that the resulting CPDA $\mathcal{C}$ recognizes $\mathcal{T}(\mathcal{L}(\mathcal{H}))$. This amounts to taking as the state set of $\mathcal{C}$ the product of state sets of $\mathcal{B}$ and $\mathcal{T}$, and appropriately combining their transitions.
- We change the word-recognizing nondeterministic CPDA $\mathcal{C}$ back to a deterministic tree-generating CPDA $\mathcal{D}$; in particular, in all configurations with multiple successors, we generate an nd-labeled node with multiple children (and in configurations with no successors, we generate an nd-labeled leaf). The CPDA $\mathcal{D}$ generates a tree T such that $\mathcal{L}(T) = \mathcal{T}(\mathcal{L}(\mathcal{H}))$.
- We translate $\mathcal{D}$ back to a recursion scheme $\mathcal{H}_\mathcal{T}$ such that $BT(\Lambda(\mathcal{H}_\mathcal{T})) = T$ (Hague et al., 2008), that is, $\mathcal{L}(\mathcal{H}_\mathcal{T}) = \mathcal{T}(\mathcal{L}(\mathcal{H}))$.
- Finally, we notice that all the modifications can be performed in polynomial time (so, in particular, $\mathcal{H}_\mathcal{T}$ is of polynomial size). Moreover, none of them increases the order, and thus the order of $\mathcal{H}_\mathcal{T}$ is at most m, as required. □

Downward closure for trees. One can also consider the downward closure of a language of trees, defined as a set of all trees that can be homeomorphically embedded in trees from the language. By Kruskal's tree theorem (Kruskal, 1960) downward closures of tree languages are regular languages of trees. We notice, however, that (unlike for words) an algorithm solving SUP is highly insufficient for the purpose of computing the downward closure. Even in the single-letter case, in order to compute $L{\downarrow}$, one has to check,

[(vii)] Definitions of finite-state transducers and collapsible pushdown automata are omitted here. We describe our procedure only on a high level of abstraction, so details of these definitions are actually irrelevant for us. It is standard to adapt the constructions proposed here to concrete formal definitions.

in particular, whether for every $n \in \mathbb{N}$, a full binary tree of depth n can be embedded in some tree from L; using SUP, we can only determine whether L contains arbitrarily large trees. Extending our techniques to this kind of problems is an interesting direction for further work.

12 Conclusions

In this paper, we have developed a type system describing simultaneous unboundedness for higher-order recursion schemes (actually, for simply-typed lambda-terms). This type system allowed then to establish the complexity of the simultaneous unboundedness problem for schemes, both in the general case, and in the case of word-recognizing schemes. Additionally, we have shown the reflection property for this problem; this property is a key ingredient used while proving decidability of model-checking trees generated by schemes with respect to the WMSO+U logic (Parys, 2018b). Some directions for further work are listed in Section 11.

References

A. V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi: 10.1145/321479.321488.

K. Asada and N. Kobayashi. On word and frontier languages of unsafe higher-order grammars. In Chatzigiannakis et al. (2016), pages 111:1–111:13. ISBN 978-3-95977-013-2. doi: 10.4230/LIPIcs.ICALP.2016.111.

K. Asada and N. Kobayashi. Pumping lemma for higher-order languages. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 97:1–97:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. ISBN 978-3-95977-041-5. doi: 10.4230/LIPIcs.ICALP.2017.97.

L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi: 10.1142/S0129054196000191.

C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In S. R. D. Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. ISBN 978-3-939897-60-6. doi: 10.4230/LIPIcs.CSL.2013.129.

I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors. *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, 2016. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-013-2.

L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. Ordered tree-pushdown systems. In P. Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. ISBN 978-3-939897-97-2. doi: 10.4230/LIPIcs.FSTTCS.2015.163.

L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In M. Grohe, E. Koskinen, and N. Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. ISBN 978-1-4503-4391-6. doi: 10.1145/2933575.2934527.

W. Czerwiński, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In A. Kosowski and I. Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. ISBN 978-3-319-22176-2. doi: 10.1007/978-3-319-22177-9_14.

W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi: 10.1016/0304-3975(82)90009-3.

A. Haddad. IO vs OI in higher-order recursion schemes. In D. Miller and Z. Ésik, editors, *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012.*, volume 77 of *EPTCS*, pages 23–30, 2012. doi: 10.4204/EPTCS.77.4.

A. Haddad. Model checking and functional program transformations. HAL, 2013a. URL `https://hal.archives-ouvertes.fr/hal-00865682`.

A. Haddad. Model checking and functional program transformations. In A. Seth and N. K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 115–126. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013b. ISBN 978-3-939897-64-4. doi: 10.4230/LIPIcs.FSTTCS.2013.115.

M. Hague, A. S. Murawski, C. L. Ong, and O. Serre. 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*, pages 452–461. IEEE Computer Society, 2008. ISBN 978-0-7695-3183-0. doi: 10.1109/LICS.2008.34.

M. Hague, J. Kochems, and C. L. Ong. Unboundedness and downward closures of higher-order pushdown automata. In R. Bodík and R. Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. ISBN 978-1-4503-3549-2. doi: 10.1145/2837614.2837627.

R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. ISBN 0-306-30707-3. doi: 10.1007/978-1-4684-2001-2_9.

A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of

Lecture Notes in Computer Science, pages 566–577. Springer, 2012. ISBN 978-3-642-32588-5. doi: 10.1007/978-3-642-32589-2_50.

N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Z. Shao and B. 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*, pages 416–428. ACM, 2009. ISBN 978-1-60558-379-2. doi: 10.1145/1480881.1480933.

N. Kobayashi. Pumping by typing. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 398–407. IEEE Computer Society, 2013. ISBN 978-1-4799-0413-6. doi: 10.1109/LICS.2013.46.

N. Kobayashi and C. L. Ong. 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*, pages 179–188. IEEE Computer Society, 2009. ISBN 978-0-7695-3746-7. doi: 10.1109/LICS.2009.29.

N. Kobayashi and C. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011. doi: 10.2168/LMCS-7(4:9)2011.

N. Kobayashi, K. Inaba, and T. Tsukada. Unsafe order-2 tree languages are context-sensitive. In A. 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*, volume 8412 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2014. ISBN 978-3-642-54829-1. doi: 10.1007/978-3-642-54830-7_10.

G. M. Kobele and S. Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi: 10.1016/j.ic.2014.12.015.

J. B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960. ISSN 00029947. doi: 10.2307/1993287.

P. Parys. 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*, pages 521–530. IEEE Computer Society, 2012. ISBN 978-1-4673-2263-8. doi: 10.1109/LICS.2012.62.

P. Parys. How many numbers can a lambda-term contain? In M. Codish and E. Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2014. ISBN 978-3-319-07150-3. doi: 10.1007/978-3-319-07151-0_19.

P. Parys. A characterization of lambda-terms transforming numerals. *Journal of Functional Programming*, 26(e12), 2016. doi: 10.1017/S0956796816000113.

P. Parys. The complexity of the diagonal problem for recursion schemes. In S. V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages

45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017a. ISBN 978-3-95977-055-2. doi: 10.4230/LIPIcs.FSTTCS.2017.45.

P. Parys. Intersection types and counting. In N. Kobayashi, editor, *Proceedings Eighth Workshop on Intersection Types and Related Systems, Porto, Portugal, 26th June 2016*, volume 242 of *Electronic Proceedings in Theoretical Computer Science*, pages 48–63. Open Publishing Association, 2017b. doi: 10.4204/EPTCS.242.6.

P. Parys. Homogeneity without loss of generality. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018a. ISBN 978-3-95977-077-4. doi: 10.4230/LIPIcs.FSCD.2018.27.

P. Parys. Recursion schemes and the WMSO+U logic. In R. Niedermeier and B. Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018b. ISBN 978-3-95977-062-0. doi: 10.4230/LIPIcs.STACS.2018.53.

S. J. Ramsay, R. P. Neatherway, and C. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In S. Jagannathan and P. Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. ISBN 978-1-4503-2544-8. doi: 10.1145/2535838.2535873.

S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. *Logical Methods in Computer Science*, 11(2), 2015. doi: 10.2168/LMCS-11(2:7)2015.

S. Salvati and I. Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi: 10.1017/S0960129514000590.

G. Zetzsche. An approach to computing downward closures. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015. ISBN 978-3-662-47665-9. doi: 10.1007/978-3-662-47666-6_35.

G. Zetzsche. The complexity of downward closure comparisons. In Chatzigiannakis et al. (2016), pages 123:1–123:14. ISBN 978-3-95977-013-2. doi: 10.4230/LIPIcs.ICALP.2016.123.

A Our definition of schemes

In this section we comment on differences between definitions contained in our paper and those that appear usually.

A.1 Letters are unranked

In the context of higher-order recursion schemes one usually considers alphabets that are ranked. This means that every letter $a \in \Sigma$ has assigned a number $rank(a)$, so that every a-labeled node has $rank(a)$ children. Since our definition is less restrictive, our type system and our algorithm carry over to the situation of a ranked alphabet. On the other hand, reductions of our hardness proofs (Section 9.4) can be easily adapted to produce schemes using a ranked alphabet.

A.2 Node constructors

The usual definition of lambda-terms does not include node constructors. Instead, for every letter a of rank r one has a lambda-term a of sort $\underbrace{o \to \ldots \to o}_{r} \to o$; after applying r arguments $P_1, \ldots, P_r$ we obtain a lambda-term equivalent to our $a\langle P_1, \ldots, P_r\rangle$. There are easy translations between lambda-terms in these formalisms: $a\langle P_1, \ldots, P_r\rangle$ can be replaced by $a\, P_1\, \ldots\, P_r$, and a can be replaced by $\lambda x_1. \cdots . \lambda x_r . a\langle x_1, \ldots, x_r\rangle$; these translations preserve Böhm trees, and can be performed in logarithmic space.

A.3 Looser definition of schemes

Let us recall the classic definition of a nondeterministic recursion scheme, and of a language recognized by such a scheme. In this definition, instead of a function $\mathcal{R}$, we have a set $\mathcal{R}_{cl}$ of rules of the form $N^{\alpha_1 \to \ldots \to \alpha_s \to o}\, x_1^{\alpha_1}\, \ldots\, x_s^{\alpha_s} \to P^o$, where $N \in \mathcal{N}$ is a nonterminal, and P is a finite *applicative* term whose all free variables are contained $\mathcal{N} \cup \{x_1^{\alpha_1}, \ldots, x_s^{\alpha_s}\}$, and where the nd symbol is not used. By an applicative term we understand a lambda-term that does not contain lambda-binders. Having a scheme $\mathcal{G}_{cl}$, we define $\to_{\mathcal{G}_{cl}}$ as the smallest relation such that

- $N\, P_1\, \ldots\, P_s \to_{\mathcal{G}_{cl}} Q[P_1/x_1, \ldots, P_s/x_s]$ if $(N\, x_1\, \ldots\, x_s \to Q) \in \mathcal{R}_{cl}$, and
- $a\langle P_1, \ldots, P_r\rangle \to_{\mathcal{G}_{cl}} a\langle P_1', \ldots, P_r'\rangle$ if $P_i \to_{\mathcal{G}_{cl}} P_i'$ for some $i \in \{1, \ldots, r\}$ and $P_j = P_j'$ for all $j \in \{1, \ldots, r\} \setminus \{i\}$.

The language recognized by $\mathcal{G}_{cl}$, denoted $\mathcal{L}_{cl}(\mathcal{G}_{cl})$, contains all finite trees T such that $N_0 \to_{\mathcal{G}_{cl}}^* T$ (where N_0 is the starting nonterminal). The order of $\mathcal{G}_{cl}$ is defined as the maximum of orders of its nonterminals.

Proposition A.1. *For every scheme $\mathcal{G}_{cl}$ understood in the classic sense one can construct in logarithmic space a scheme $\mathcal{G}$ that sticks to our definition, is of the same order, and such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{cl}(\mathcal{G}_{cl})$.*

Proof: Consider a scheme $\mathcal{G}_{cl} = (\mathcal{N}, \mathcal{R}_{cl}, N_0)$ understood in the classic sense. Out of it, we construct a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$ sticking to our definition: for every nonterminal N we consider all rules $(N\, x_1\, \ldots\, x_s \to P_1), \ldots, (N\, x_1\, \ldots\, x_s \to P_m)$ of $\mathcal{G}_{cl}$ concerning this nonterminal, and we take $\mathcal{R}(N) = \lambda x_1. \cdots . \lambda x_s . \mathsf{nd}\langle P_1, \ldots, P_m\rangle$. Notice that $\mathcal{R}(N)$ never equals a nonterminal, as required by our definition. Clearly this translation preserves the order of the scheme and can be performed in logarithmic space. We now show that $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{cl}(\mathcal{G}_{cl})$.

Consider a finite Σ-labeled tree $T = a\langle T_1, \ldots, T_r\rangle$, and a finite applicative term P with free variables in $\mathcal{N}$, and not using the nd symbol. We are going to prove by induction on $n + |T|$ that $P \to_{\mathcal{G}_{cl}}^n T$ if and only

if $BT(\Lambda_{\mathcal{G}}(P)) \to_{\mathsf{nd}}^n T$. This equivalence implies that $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{cl}(\mathcal{G}_{cl})$, since by definition $\mathcal{L}(\mathcal{G})$ contains finite Σ-labeled trees T such that $BT(\Lambda_{\mathcal{G}}(N_0)) \to_{\mathsf{nd}}^n T$ for some $n \in \mathbb{N}$, while $\mathcal{L}_{cl}(\mathcal{G}_{cl})$ contains finite trees T such that $N_0 \to_{\mathcal{G}_{cl}}^n T$ for some $n \in \mathbb{N}$ (all the latter trees are also Σ-labeled since $\mathcal{G}_{cl}$ does not use the nd symbol). There are two possible shapes of P. Suppose first that $P = N\,Q_1\,\dots\,Q_s$, where N is a nonterminal. Let $(N\,x_1\,\dots\,x_s \to R_1), \dots, (N\,x_1\,\dots\,x_s \to R_m)$ be all rules of $\mathcal{G}_{cl}$ concerning N. We have that $P \to_{\mathcal{G}_{cl}}^n T$ if and only if $P = N\,Q_1\,\dots\,Q_s \to_{\mathcal{G}_{cl}} R_i[Q_1/x_1, \dots, Q_s/x_s] \to_{\mathcal{G}_{cl}}^{n-1} T$ for some $i \in \{1, \dots, m\}$. On the other hand,

$$\begin{aligned}\Lambda_{\mathcal{G}}(P) &= (\lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle\Lambda_{\mathcal{G}}(R_1), \dots, \Lambda_{\mathcal{G}}(R_m)\rangle)\,\Lambda_{\mathcal{G}}(Q_1)\,\dots\,\Lambda_{\mathcal{G}}(Q_s)\\ &\xrightarrow{h,s}_{\beta} \mathsf{nd}\langle\Lambda_{\mathcal{G}}(R_1[Q_1/x_1, \dots, Q_s/x_s]), \dots, \Lambda_{\mathcal{G}}(R_m[Q_1/x_1, \dots, Q_s/x_s])\rangle\,.\end{aligned}$$

Thus $BT(\Lambda_{\mathcal{G}}(P)) \to_{\mathsf{nd}}^n T$ if and only if

$$BT(\Lambda_{\mathcal{G}}(P)) \to_{\mathsf{nd}} BT(\Lambda_{\mathcal{G}}(R_i[Q_1/x_1, \dots, Q_s/x_s])) \to_{\mathsf{nd}}^{n-1} T \qquad \text{for some } i \in \{1, \dots, m\}\,.$$

We have $R_i[Q_1/x_1, \dots, Q_s/x_s] \to_{\mathcal{G}_{cl}}^{n-1} T$ if and only if $BT(\Lambda_{\mathcal{G}}(R_i[Q_1/x_1, \dots, Q_s/x_s])) \to_{\mathsf{nd}}^{n-1} T$ by the induction assumption.

The other possible case is that $P = b\langle P_1, \dots, P_k\rangle$, where $b \neq \mathsf{nd}$. Then $P \to_{\mathcal{G}_{cl}}^n T$ if and only if $b = a$, $k = r$, and $P_i \to_{\mathcal{G}_{cl}}^{n_i} T_i$ for all $i \in \{1, \dots, r\}$, where $n = n_1 + \dots + n_r$. On the other hand, $BT(\Lambda_{\mathcal{G}}(P)) \to_{\mathsf{nd}}^n T$ if and only if $b = a$, $k = r$, and $BT(\Lambda_{\mathcal{G}}(P_i)) \to_{\mathsf{nd}}^{n_i} T_i$ for all $i \in \{1, \dots, r\}$, where $n = n_1 + \dots + n_r$. We have $P_i \to_{\mathcal{G}_{cl}}^{n_i} T_i$ if and only if $BT(\Lambda_{\mathcal{G}}(P_i)) \to_{\mathsf{nd}}^{n_i} T_i$ by the induction assumption. This finishes the proof of the equality $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{cl}(\mathcal{G}_{cl})$. □

Due to the above translation, our algorithm can be applied to schemes conforming to the classic definition. On the other hand, it is easy to modify our hardness proofs (cf. Section 9.4) so that the reductions used there will produce schemes conforming to the classic definition, which will show hardness also for such schemes.[(viii)]

B Proof of Lemma 10.3

In this section we recall from Haddad (2012, Section 4.2) the construction of a scheme $\mathcal{G}_{\mathcal{M}}$ generating a $\Xi^m_{\mathcal{M}}$-reflection of a given scheme $\mathcal{G}$, and we prove its correctness. We need to adapt the construction slightly, because Haddad uses the more restrictive definition of recursion schemes described in Appendix A.3 (without lambda-binders inside right sides of rules), because he allows non-homogeneous sorts, and because he uses constants instead of node constructors.

For the rest of this section we fix a number $m \in \mathbb{N}$, a morphism $\mathcal{M}$ to a finitary applicative structure D, and a recursion scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$ of order at most m (note that here, unlike in most sections of this paper, the order of $\mathcal{G}$ is at most m, not at most $m + 1$).

B.1 Transformation of the scheme

For every sort $\alpha = \alpha_1 \to \dots \to \alpha_s \to o$ let $Arg^\alpha = D[\alpha_1] \times \dots \times D[\alpha_s]$; fix also some (arbitrary) linear order $\preceq$ on elements of Arg^α, for every α. Moreover, for every lambda-term P, let $Free^{\uparrow}(P)$ be the set of lists of the form $[(x_1, \chi_1), \dots, (x_k, \chi_k)]$, where the variables $x_1, \dots, x_k \in Vars \setminus \mathcal{N}$ are pairwise distinct,

(viii) A translation in the opposite direction is also possible, but we do not give it here, as it is more technical.

where $\chi_i \in D[\alpha_i]$ for α_i being the sort of x_i, for all $i \in \{1, \ldots, k\}$, and where every free variable of P not being a nonterminal appears as x_i for some i. Furthermore, let $\mathit{Free}(P) \subseteq \mathit{Free}^{\uparrow}(P)$ contain only those lists, where every variable x_i is free in P. Given a list $\nu \in \mathit{Free}^{\uparrow}(P)$, by $\nu\restriction_P$ we denote the sublist in $\mathit{Free}(P)$ obtained from ν by removing all pairs (x, χ) with x not being free in P. Elements of Arg^{α} and $\mathit{Free}(P)$ are used to store values of the morphism for all arguments of a lambda-term P of sort α, and for all its free variables (other than nonterminals), respectively.

Let $\mathbf{\Lambda}_m$ be the set of all lambda-terms that

- are of complexity at most m,
- whose all free variables are of order at most $m-1$, and
- such that in every its subterm there are only finitely many free occurrences of every variable.

In particular, every subterm of $\Lambda(\mathcal{G})$ is in $\mathbf{\Lambda}_m$. Moreover, all subterms of lambda-terms from $\mathbf{\Lambda}_m$ are in $\mathbf{\Lambda}_m$ as well.

Next, we need to extend the morphism $\mathcal{M}$ to lambda-terms that are not closed. For a lambda-term $P \in \mathbf{\Lambda}_m$, and for a list $\nu = [(x_1, \chi_1), \ldots, (x_k, \chi_k)] \in \mathit{Free}(P)$, we would like to define the value of P as $[\![\lambda x_1.\cdots.\lambda x_k.P]\!]_{\mathcal{M}}\, \chi_1\, \ldots\, \chi_k$. Such a definition is, however, invalid, because the sort of $\lambda x_1.\cdots.\lambda x_k.P$ does not need to be homogeneous (although we may list the variables $x_1, \ldots, x_k$ starting from those of higher order, P may require arguments of order higher than the order of x_k), while $\mathcal{M}$ is defined only for lambda-terms having homogeneous sorts. For this reason, we artificially raise the order of free variables to $m-1$, as described below.

Our definitions depend on the value of m.

- Suppose first that $m \geq 2$. In this case, we fix a lambda-term Z_{m-2} and a variable z_{m-2}, both of the same sort $\alpha^{\mathsf{z}}_{m-2}$ such that $\mathit{ord}(\alpha^{\mathsf{z}}_{m-2}) = m-2$. To be concrete, we take $\alpha^{\mathsf{z}}_0 = o$ and $\alpha^{\mathsf{z}}_i = \alpha^{\mathsf{z}}_{i-1} \to o$ for all $i \geq 1$, and then we take $Z_{m-2} = \mathsf{a}\langle\rangle$ if $m-2 = 0$ and $Z_{m-2} = \lambda \mathsf{z}_{m-3}.\mathsf{a}\langle\rangle$ for z_{m-3} of sort $\alpha^{\mathsf{z}}_{m-3}$ if $m-2 \geq 1$ (where a is a fixed letter). Next, for every sort α of order at most $m-1$ we define $\mathit{up}(\alpha) = \alpha^{\mathsf{z}}_{m-2} \to \alpha$, for every lambda-term Q of sort α we define $\mathit{up}(Q) = \lambda z_{m-2}.Q$, and for every lambda-term R of sort $\mathit{up}(\alpha)$ we define $\mathit{down}(R) = R\, Z_{m-2}$.
- If $m = 1$, we instead define $\mathit{up}(\alpha) = \alpha$, and $\mathit{up}(Q) = Q$, and $\mathit{down}(R) = R$.
- If $m = 0$, there are no sorts of order at most $m-1$, so the definitions become useless.

For a lambda-term Q of sort α with $\mathit{ord}(\alpha) \leq m-1$ we observe that $\mathit{up}(Q)$ is of sort $\mathit{up}(\alpha)$, and $\mathit{ord}(\mathit{up}(\alpha)) = m-1$, and $\mathit{down}(\mathit{up}(Q)) \xrightarrow{h}{}^{*}_{\beta} Q$. One may say that $\mathit{up}(\cdot)$ raises the order of a lambda-term Q to $m-1$, and $\mathit{down}(\cdot)$ decreases it back to the original order.

Going further, for every variable x of sort α, where $\mathit{ord}(\alpha) \leq m-1$, we fix a corresponding variable $\overline{x}$ of sort $\mathit{up}(\alpha)$, assuming that this mapping is injective (i.e., that $\overline{x} \neq \overline{y}$ when $x \neq y$).

Moreover, for every element $\chi \in D[\alpha]$, where $\mathit{ord}(\alpha) \leq m-1$, let $\mathit{up}(\chi) = \boldsymbol{\lambda}\alpha^{\mathsf{z}}_{m-2}.\chi$ if $m \geq 2$, and let $\mathit{up}(\chi) = \chi$ if $m = 1$. Observe that $\mathit{up}([\![Q]\!]_{\mathcal{M}}) = [\![\mathit{up}(Q)]\!]_{\mathcal{M}}$ for every closed lambda-term Q of sort α.

We now already have all ingredients needed to extend our morphism to lambda-terms that are not closed. Thus, for a lambda-term $P \in \mathbf{\Lambda}_m$, and for a list $\nu = [(x_1, \chi_1), \ldots, (x_k, \chi_k)] \in \mathit{Free}(P)$, we define $[\![P]\!]^{\nu}_{\mathcal{M}} = [\![\lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.P[\mathit{down}(\overline{x_1})/x_1, \ldots, \mathit{down}(\overline{x_k})/x_k]]\!]_{\mathcal{M}}\, \mathit{up}(\chi_1)\, \ldots\, \mathit{up}(\chi_k)$. Notice that if P is closed, then the list ν is empty, and, in consequence, $[\![P]\!]^{\nu}_{\mathcal{M}}$ is just $[\![P]\!]_{\mathcal{M}}$.

Example B.1. Suppose that $m = 2$, and consider a lambda-term $P = \lambda \mathsf{y}.\mathsf{y}\,\mathsf{x}$, where y is of sort $o \to o$, and x is of sort o. Consider also a list $\nu = (\mathsf{x}, \chi)$, where $\chi = [\![\mathsf{b}\langle\rangle]\!]_{\mathcal{M}}$. In this case $[\![P]\!]^{\nu}_{\mathcal{M}} = [\![\lambda\overline{\mathsf{x}}.\lambda\mathsf{y}.\mathsf{y}\,(\overline{\mathsf{x}}\,\mathsf{a}\langle\rangle)]\!]_{\mathcal{M}}\, \mathit{up}([\![\mathsf{b}\langle\rangle]\!]_{\mathcal{M}}) = [\![(\lambda\overline{\mathsf{x}}.\lambda\mathsf{y}.\mathsf{y}\,(\overline{\mathsf{x}}\,\mathsf{a}\langle\rangle))\,(\lambda\mathsf{z}_0.\mathsf{b}\langle\rangle)]\!]_{\mathcal{M}}$; the variables $\overline{\mathsf{x}}$ and z_0 are of sort $o \to o$ and o, respectively. □

For a finite lambda-term $R \in \mathbf{\Lambda}_m$ of sort α (possibly containing nonterminals), and for $\zeta \in \mathit{Arg}^\alpha$ and $\nu \in \mathit{Free}(R)$, we define a lambda-term $\theta(R,\zeta,\nu)$ of sort $\alpha^\bullet$, where sorts $\alpha^\bullet$ are defined by induction:

$$o^\bullet = o \qquad \text{and} \qquad (\beta \to \gamma)^\bullet = \underbrace{\beta^\bullet \to \ldots \to \beta^\bullet}_{|\mathit{Arg}^\beta|} \to \gamma^\bullet .$$

The translation is defined by induction on the structure of R as follows:

- if $R = a\langle P_1, \ldots, P_r\rangle$, let $\theta(R,\zeta,\nu) = (a, [\![\Lambda_{\mathcal{G}}(R)]\!]^\nu_{\mathcal{M}})\langle \theta(P_1,\zeta,\nu{\restriction_{P_1}}), \ldots, \theta(P_r,\zeta,\nu{\restriction_{P_r}})\rangle$;
- if $R = x^\alpha$ (including the case when it is a nonterminal), let $\theta(R,\zeta,\nu) = x^{\alpha^\bullet}_\zeta$;
- if $R = P^{\beta\to\alpha}\,Q^\beta$, let $\theta(R,\zeta,\nu) = \theta(P, [\![\Lambda_{\mathcal{G}}(Q)]\!]^{\nu{\restriction_Q}}_{\mathcal{M}} :: \zeta, \nu{\restriction_P})\,\theta(Q,\xi_1,\nu{\restriction_Q})\,\ldots\,\theta(Q,\xi_n,\nu{\restriction_Q})$, where $\xi_1,\ldots,\xi_n$ are all the elements of Arg^β, ordered by $\preceq$;
- if $R = \lambda x^\beta.P$ and $\zeta = \chi :: \zeta'$, let $\theta(R,\zeta,\nu) = \lambda x^{\beta^\bullet}_{\xi_1}.\cdots.\lambda x^{\beta^\bullet}_{\xi_n}.\theta(P,\zeta',\nu::(x^\beta,\chi){\restriction_P})$, where $\xi_1,\ldots,\xi_n$ are all the elements of Arg^β, ordered by $\preceq$.

To the resulting scheme $\mathcal{G}_{\mathcal{M}}$ we take a nonterminal $N^{\alpha^\bullet}_\zeta$ for every nonterminal N^α of $\mathcal{G}$ and every $\zeta \in \mathit{Arg}^\alpha$. In particular, for the starting nonterminal N^o_0 of $\mathcal{G}$ this results in $N^o_{0,()}$ (notice that the only element of Arg^o is $()$), which we take as the starting nonterminal of $\mathcal{G}_{\mathcal{M}}$. The rules are defined by $\mathcal{R}_{\mathcal{M}}(N^{\alpha^\bullet}_\zeta) = \theta(\mathcal{R}(N^\alpha),\zeta,[\,])$.

B.2 Correctness proof

Once the resulting scheme $\mathcal{G}_{\mathcal{M}}$ is defined, we need to prove that it indeed generates a $\Xi^m_{\mathcal{M}}$-reflection of $BT(\Lambda(\mathcal{G}))$. To this end, we need to relate lambda-terms obtained while reducing $\Lambda(\mathcal{G})$ to lambda-terms obtained while reducing $\Lambda(\mathcal{G}_{\mathcal{M}})$. We now define an appropriate relation.

First, for every lambda-term $P \in \mathbf{\Lambda}_m$, and for every list $\nu = [(x_1,\chi_1),\ldots,(x_k,\chi_k)] \in \mathit{Free}(P)$, we let $\mathit{Val}(P,\nu)$ to be the set of values $[\![Q]\!]_{\mathcal{M}}\ \mathit{up}(\chi_1)\ \ldots\ \mathit{up}(\chi_k)$ over all closed lambda-terms Q of complexity at most m, such that $Q \to^*_\beta \lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.P[\mathit{down}(\overline{x_1})/x_1,\ldots,\mathit{down}(\overline{x_k})/x_k]$. Notice that elements of $\mathit{Val}(P,\nu)$ are similar to $[\![P]\!]^\nu_{\mathcal{M}}$, with the exception that in the definition we allow to replace $\lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.P[\mathit{down}(\overline{x_1})/x_1,\ldots,\mathit{down}(\overline{x_k})/x_k]$ by an arbitrary closed lambda-term Q of complexity at most m, such that $Q \to^*_\beta \lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.P[\mathit{down}(\overline{x_1})/x_1,\ldots,\mathit{down}(\overline{x_k})/x_k]$. In particular, by taking $Q = \lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.P[\mathit{down}(\overline{x_1})/x_1,\ldots,\mathit{down}(\overline{x_k})/x_k]$ we obtain the following lemma.

Lemma B.1. *For every lambda-term $P \in \mathbf{\Lambda}_m$ and for every $\nu \in \mathit{Free}(P)$ it is the case that $[\![P]\!]^\nu_{\mathcal{M}} \in \mathit{Val}(P,\nu)$.* □

Next, for a lambda-term $R \in \mathbf{\Lambda}_m$ of sort α, for $\zeta \in \mathit{Arg}^\alpha$, and for $\nu \in \mathit{Free}(R)$, we define a set $\Theta(R,\zeta,\nu)$ of lambda-terms of sort $\alpha^\bullet$, by coinduction on the structure of R:

- if $R = a\langle P_1,\ldots,P_r\rangle$, then $\Theta(R,\zeta,\nu)$ contains lambda-terms $(a,\chi)\langle S_1,\ldots,S_r\rangle$ such that $\chi \in \mathit{Val}(R,\nu)$ and $S_i \in \Theta(P_i,\zeta,\nu{\restriction_{P_i}})$ for $i \in \{1,\ldots,r\}$;
- if $R = x^\alpha$, then $\Theta(R,\zeta,\nu) = \{x^{\alpha^\bullet}_\zeta\}$;
- if $R = P^{\beta\to\alpha}\,Q^\beta$, then $\Theta(R,\zeta,\nu)$ contains lambda-terms $S\,U_1\ \ldots\ U_n$ such that $S \in \Theta(P, \chi :: \zeta, \nu{\restriction_P})$ for some $\chi \in \mathit{Val}(Q,\nu{\restriction_Q})$, and $U_i \in \Theta(Q,\xi_i,\nu{\restriction_Q})$ for all $i \in \{1,\ldots,n\}$, where $\xi_1,\ldots,\xi_n$ are all the elements of Arg^β, ordered by $\preceq$;
- if $R = \lambda x^\beta.P$ and $\zeta = \chi :: \zeta'$, then $\Theta(R,\zeta,\nu)$ contains lambda-terms $\lambda x^{\beta^\bullet}_{\xi_1}.\cdots.\lambda x^{\beta^\bullet}_{\xi_n}.S$ such that $S \in \Theta(P,\zeta',\nu::(x^\beta,\chi){\restriction_P})$, where $\xi_1,\ldots,\xi_n$ are all the elements of Arg^β, ordered by $\preceq$.

Easy coinduction shows that elements of $\Theta(R, \zeta, \nu)$ have the same free variables as R, up to an appropriate renaming.

Lemma B.2. *If $R' \in \Theta(R, \zeta, \nu)$ for some $R^\alpha \in \mathbf{\Lambda}_m$, $\zeta \in Arg^\alpha$, and $\nu \in Free(R)$, and if z^β is not free in R, then $z^{\beta\bullet}_\xi$ is not free in R' for any $\xi \in Arg^\beta$.* □

In the next lemma we observe that θ is a special case of Θ.

Lemma B.3. *Let R be a finite lambda-term of sort α, let $\zeta \in Arg^\alpha$, and let $\nu \in Free(R)$. Then $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) \in \Theta(\Lambda_\mathcal{G}(R), \zeta, \nu)$.*

Notice that R and $\Lambda_\mathcal{G}(R)$ have the same free variables (except for nonterminals used in R), and thus we have $\nu \in Free(\Lambda_\mathcal{G}(R))$. We use the same observation in the proof below, while saying that $\nu{\restriction_P}$ equals $\nu{\restriction_{\Lambda_\mathcal{G}(P)}}$, for any subterm P of R.

Proof of Lemma B.3: The proof is by coinduction on the structure of $\Lambda_\mathcal{G}(R)$. Suppose first that $R = N^\alpha$ is a nonterminal. Then we have $\Lambda_\mathcal{G}(R) = \Lambda_\mathcal{G}(\mathcal{R}(N^\alpha))$ and $\nu = [\,]$. Observe that $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) = \Lambda_{\mathcal{G}_\mathcal{M}}(N^{\alpha\bullet}_\zeta) = \Lambda_{\mathcal{G}_\mathcal{M}}(\mathcal{R}_\mathcal{M}(N^{\alpha\bullet}_\zeta)) = \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(\mathcal{R}(N^\alpha), \zeta, [\,]))$. We can thus equally well consider $\mathcal{R}(N^\alpha)$ instead of R. Recalling that $\mathcal{R}(N^\alpha)$ cannot be equal to a nonterminal, we have reduced this case to the case when R is not a nonterminal. Thus, for the remaining part of the proof we suppose that R is not a nonterminal.

We have four cases depending on the shape of R. Suppose first that $R = a\langle P_1, \ldots, P_r\rangle$. Then $\theta(R, \zeta, \nu) = (a, [\![\Lambda_\mathcal{G}(R)]\!]^\nu_\mathcal{M})\langle\theta(P_1, \zeta, \nu{\restriction_{P_1}}), \ldots, \theta(P_r, \zeta, \nu{\restriction_{P_r}})\rangle$, and thus

$$\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) = (a, [\![\Lambda_\mathcal{G}(R)]\!]^\nu_\mathcal{M})\langle\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P_1, \zeta, \nu{\restriction_{P_1}})), \ldots, \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P_r, \zeta, \nu{\restriction_{P_r}}))\rangle\,.$$

On the other hand, $\Lambda_\mathcal{G}(R) = a\langle\Lambda_\mathcal{G}(R_1), \ldots, \Lambda_\mathcal{G}(R_r)\rangle$. By the assumption of coinduction we obtain that $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P_i, \zeta, \nu{\restriction_{P_i}})) \in \Theta(\Lambda_\mathcal{G}(P_i), \zeta, \nu{\restriction_{P_i}})$ for all $i \in \{1, \ldots, r\}$, and due to Lemma B.1 we have that $[\![\Lambda_\mathcal{G}(R)]\!]^\nu_\mathcal{M} \in Val(\Lambda_\mathcal{G}(R), \nu)$. Thus, from the definition of Θ we can deduce that $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) \in \Theta(\Lambda_\mathcal{G}(R), \zeta, \nu)$.

Next, suppose that $R = x^\alpha$. Recall that R is not a nonterminal, thus we have $\Lambda_\mathcal{G}(R) = x^\alpha$ and $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(x^\alpha, \zeta, \nu)) = \Lambda_{\mathcal{G}_\mathcal{M}}(x^{\alpha\bullet}_\zeta) = x^{\alpha\bullet}_\zeta$. It follows from the definition of Θ that $x^{\alpha\bullet}_\zeta \in \Theta(x^\alpha, \zeta, \nu)$.

Suppose now that $R = P\,Q$. Let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$, where β is the sort of Q. Then $\theta(R, \zeta, \nu) = \theta(P, [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P})\,\theta(Q, \xi_1, \nu{\restriction_Q})\ \ldots\ \theta(Q, \xi_n, \nu{\restriction_Q})$, and thus

$$\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) = \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P, [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P}))\,\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(Q, \xi_1, \nu{\restriction_Q}))\ \ldots\ \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(Q, \xi_n, \nu{\restriction_Q}))\,.$$

On the other hand, $\Lambda_\mathcal{G}(R) = \Lambda_\mathcal{G}(P)\,\Lambda_\mathcal{G}(Q)$. By the assumption of coinduction we obtain that

$$\begin{aligned} \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P, [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P})) \in \Theta(\Lambda_\mathcal{G}(P), [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P}) \qquad &\text{and} \\ \Lambda_{\mathcal{G}_\mathcal{M}}(\theta(Q, \xi_i, \nu{\restriction_Q})) \in \Theta(\Lambda_\mathcal{G}(Q), \xi_i, \nu{\restriction_Q}) \qquad &\text{for all } i \in \{1, \ldots, n\}, \end{aligned}$$

and due to Lemma B.1 we have that $[\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} \in Val(\Lambda_\mathcal{G}(Q), \nu{\restriction_Q})$. Thus, from the definition of Θ we can deduce that $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) \in \Theta(\Lambda_\mathcal{G}(R), \zeta, \nu)$.

Finally, suppose that $R = \lambda x^\beta.P$. Again, let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$. Let also $\zeta = \chi :: \zeta'$. Then $\theta(R, \zeta, \nu) = \lambda x^{\beta\bullet}_{\xi_1}.\cdots.\lambda x^{\beta\bullet}_{\xi_n}.\theta(P, \zeta', \nu ::(x^\beta, \chi){\restriction_P})$, and thus

$$\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) = \lambda x^{\beta\bullet}_{\xi_1}.\cdots.\lambda x^{\beta\bullet}_{\xi_n}.\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P, \zeta', \nu ::(x^\beta, \chi){\restriction_P}))\,.$$

On the other hand, $\Lambda_{\mathcal{G}}(R) = \lambda x^\beta.\Lambda_{\mathcal{G}}(P)$. By the assumption of coinduction we obtain that

$$\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(P, \zeta', \nu :: (x^\beta, \chi)\restriction_P)) \in \Theta(\Lambda_{\mathcal{G}}(P), \zeta', \nu :: (x^\beta, \chi)\restriction_P)\,.$$

Thus, from the definition of Θ we can deduce that $\Lambda_{\mathcal{G}_\mathcal{M}}(\theta(R, \zeta, \nu)) \in \Theta(\Lambda_{\mathcal{G}}(R), \zeta, \nu)$. □

Next, we prove that the relation Θ is preserved during head beta-reductions (Lemma B.6). Before that, we need auxiliary lemmata for substitution (Lemmata B.4 and B.5).

Lemma B.4. *Let $R^\alpha, S^\gamma \in \mathbf{\Lambda}_m$, where S is closed, let z^γ be a variable, let $\nu \in \mathit{Free}(R[S/z])$, and let $\chi \in \mathit{Val}(S, [\,])$. Then $\mathit{Val}(R, (z, \chi) :: \nu\restriction_R) \subseteq \mathit{Val}(R[S/z], \nu)$.*

Proof: If z is not free in R, we have $R[S/z] = R$ and $(z, \chi) :: \nu\restriction_R = \nu$, which immediately gives the thesis. In the sequel we assume that z is free in R; then, in particular, $(z, \chi) :: \nu\restriction_R = (z, \chi) :: \nu$. Denote $\nu = [(x_1, \chi_1), \dots, (x_k, \chi_k)]$. Take some element of $\mathit{Val}(R, (z, \chi) :: \nu)$; by definition it is of the form $[\![P]\!]_\mathcal{M}\ up(\chi)\ up(\chi_1)\ \dots\ up(\chi_k)$ for some closed lambda-term P of complexity at most m, such that

$$P \to_\beta^* \lambda\overline{z}.\lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.R[\mathit{down}(\overline{z})/z, \mathit{down}(\overline{x_1})/x_1, \dots, \mathit{down}(\overline{x_k})/x_k]\,.$$

Moreover, because $\chi \in \mathit{Val}(S, [\,])$, there is a closed lambda-term Q of complexity at most m, such that $Q \to_\beta^* S$ and $\chi = [\![Q]\!]_\mathcal{M}$. From the definition of $up(\cdot)$ it follows that $up(Q) \to_\beta^* up(S)$ and $up(\chi) = [\![up(Q)]\!]_\mathcal{M}$. We have that

$$\begin{aligned} P\ up(Q) &\to_\beta^* (\lambda\overline{z}.\lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.R[\mathit{down}(\overline{z})/z, \mathit{down}(\overline{x_1})/x_1, \dots, \mathit{down}(\overline{x_k})/x_k])\ up(S) \\ &\to_\beta \lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.R[\mathit{down}(up(S))/z, \mathit{down}(\overline{x_1})/x_1, \dots, \mathit{down}(\overline{x_k})/x_k] \\ &\to_\beta^* \lambda\overline{x_1}.\cdots.\lambda\overline{x_k}.R[S/z][\mathit{down}(\overline{x_1})/x_1, \dots, \mathit{down}(\overline{x_k})/x_k]\,. \end{aligned}$$

The last relation holds because, by definition of $\mathbf{\Lambda}_m$, there are only finitely many free occurrences of z in R. Because $\mathcal{M}$ is a morphism, we have that $[\![P]\!]_\mathcal{M}\ [\![up(Q)]\!]_\mathcal{M} = [\![P\ up(Q)]\!]_\mathcal{M}$, and, in consequence, $[\![P]\!]_\mathcal{M}\ up(\chi)\ up(\chi_1)\ \dots\ up(\chi_k) = [\![P\ up(Q)]\!]_\mathcal{M}\ up(\chi_1)\ \dots\ up(\chi_k) \in \mathit{Val}(R[S/z], \nu)$, as needed. Notice that in this lemma it is rather important that the pair (z, χ) is the first element of the list $(z, \chi) :: \nu$. □

Lemma B.5. *Let $R^\alpha, S^\gamma \in \mathbf{\Lambda}_m$, where S is closed, let z^γ be a variable, let $\zeta \in \mathit{Arg}^\alpha$, let $\nu \in \mathit{Free}(R[S/z^\gamma])$, and let $\chi' \in \mathit{Val}(S, [\,])$. Let $\xi'_1, \dots, \xi'_m$ be all the elements of Arg^γ, ordered by $\preceq$. Let also $R' \in \Theta(R, \zeta, (z^\gamma, \chi') :: \nu\restriction_R)$, and let $S_j \in \Theta(S, \xi'_j, [\,])$ for $j \in \{1, \dots, m\}$. Then $R'[S_1/z^{\gamma\bullet}_{\xi'_1}] \dots [S_m/z^{\gamma\bullet}_{\xi'_m}] \in \Theta(R[S/z^\gamma], \zeta, \nu)$ holds.*

Proof: To shorten the notation, denote $\eta(P) = P[S_1/z^{\gamma\bullet}_{\xi'_1}] \dots [S_m/z^{\gamma\bullet}_{\xi'_m}]$ for any lambda-term P.

The proof is by coinduction on the structure of R. Observe first that if z^γ is not free in R, then $\eta(R') = R'$ (the variables $z^{\gamma\bullet}_{\xi'_j}$ are not free in R' by Lemma B.2), and $R[S/z^\gamma] = R$, and $(z^\gamma, \chi') :: \nu\restriction_R = \nu$, which immediately gives the thesis. In the sequel we assume that z^γ is free in R; then in particular $(z^\gamma, \chi') :: \nu\restriction_R = (z^\gamma, \chi') :: \nu$. We have four cases depending on the shape of R.

Suppose first that $R = a\langle P_1, \dots, P_r\rangle$. Then $R' \in \Theta(R, \zeta, (z^\gamma, \chi') :: \nu)$ implies by the definition of Θ that R' is of the form $(a, \chi)\langle Q_1, \dots, Q_r\rangle$, where $\chi \in \mathit{Val}(R, (z^\gamma, \chi') :: \nu)$ and $Q_i \in \Theta(P_i, \zeta,$

$(z^\gamma, \chi') :: \nu\restriction_{P_i})$ for $i \in \{1, \ldots, r\}$. We have that $\chi \in \mathit{Val}(R[S/z^\gamma], \nu)$ by Lemma B.4, and $\eta(Q_i) \in \Theta(P_i[S/z^\gamma], \zeta, \nu\restriction_{P_i})$ for all $i \in \{1, \ldots, r\}$ by the assumption of coinduction. Thus, by the definition of Θ,

$$\eta(R') = (a, \chi)\langle \eta(Q_1), \ldots, \eta(Q_r)\rangle \in \Theta(a\langle P_1[S/z^\gamma], \ldots, P_r[S/z^\gamma]\rangle, \zeta, \nu) = \Theta(R[S/z^\gamma], \zeta, \nu)\,.$$

Next, suppose that $R = z^\gamma$ (recall that z^γ is free in R, so R cannot be a variable other than z^γ). Then $R' \in \Theta(R, \zeta, (z^\gamma, \chi') :: \nu)$ implies by the definition of Θ that $R' = z_\zeta^{\gamma\bullet}$. Moreover, $\zeta = \xi'_j$ for some $j \in \{1, \ldots, m\}$, and $\nu = [\,]$. Because S is closed, by Lemma B.2 we have that variables $z_{\xi'_i}^{\gamma\bullet}$ are not free in S_j. Thus, $\eta(R') = S_j[S_{j+1}/z_{\xi'_{j+1}}^{\gamma\bullet}] \ldots [S_m/z_{\xi'_m}^{\gamma\bullet}] = S_j \in \Theta(S, \xi'_j, [\,]) = \Theta(R[S/z^\gamma], \zeta, \nu)$.

Suppose now that $R = P\,Q$. Let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$, where β is the sort of Q. Then $R' \in \Theta(R, \zeta, (z^\gamma, \chi') :: \nu)$ implies by the definition of Θ that R' is of the form $P'\,Q_1\,\ldots\,Q_n$, where $P' \in \Theta(P, \chi :: \zeta, (z^\gamma, \chi') :: \nu\restriction_P)$ for some $\chi \in \mathit{Val}(Q, (z^\gamma, \chi') :: \nu\restriction_Q)$, and $Q_i \in \Theta(Q, \xi_i, (z^\gamma, \chi') :: \nu\restriction_Q)$ for all $i \in \{1, \ldots, n\}$. We have that $\eta(P') \in \Theta(P[S/z^\gamma], \chi :: \zeta, \nu\restriction_P)$ and $\eta(Q_i) \in \Theta(Q[S/z^\gamma], \xi_i, \nu\restriction_Q)$ for all $i \in \{1, \ldots, n\}$ by the assumption of coinduction, and $\chi \in \mathit{Val}(Q[S/z^\gamma], \nu\restriction_Q)$ by Lemma B.4. Thus,

$$\eta(R') = \eta(P')\,\eta(Q_1)\,\ldots\,\eta(Q_n) \in \Theta(P[S/z^\gamma]\,Q[S/z^\gamma], \zeta, \nu) = \Theta(R[S/z^\gamma], \zeta, \nu)$$

by the definition of Θ.

Finally, suppose that $R = \lambda x^\beta.P$. Let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$, and let $\zeta = \chi :: \zeta'$. Since z^γ is free in R, we have $x^\beta \neq z^\gamma$. Then $R' \in \Theta(R, \zeta, (z^\gamma, \chi') :: \nu)$ implies by the definition of Θ that $R' = \lambda x_{\xi_1}^{\beta\bullet}.\cdots.\lambda x_{\xi_n}^{\beta\bullet}.Q$, where $Q \in \Theta(P, \zeta', (z^\gamma, \chi') :: \nu :: (x^\beta, \chi)\restriction_P)$. By the assumption of coinduction we obtain that $\eta(Q) \in \Theta(P[S/z^\gamma], \zeta', \nu :: (x^\beta, \chi)\restriction_P)$, so $\eta(R') = \lambda x_{\xi_1}^{\beta\bullet}.\cdots.\lambda x_{\xi_n}^{\beta\bullet}.\eta(Q) \in \Theta(\lambda x^\beta.P[S/z^\gamma], \zeta, \nu) = \Theta(R[S/z^\gamma], \zeta, \nu)$ by the definition of Θ. □

Lemma B.6. *Let $P, Q \in \mathbf{\Lambda}_m$ be closed lambda-terms of sort o such that $P \xrightarrow{h}_\beta Q$. If $U \in \Theta(P, (), [\,])$, then there exists $V \in \Theta(Q, (), [\,])$ such that $U \xrightarrow{h}_\beta^+ V$.*

Proof: The condition $P \xrightarrow{h}_\beta Q$ implies that P is of the form $(\lambda x^{\alpha_0}.R)\,S_0\,S_1\,\ldots\,S_s$, and then $Q = R[S_0/x^{\alpha_0}]\,S_1\,\ldots\,S_s$. For $i \in \{0, \ldots, s\}$, let α_i be the sort of S_i, and let $\xi_{i,1}, \ldots, \xi_{i,n_i}$ be all the elements of Arg^{α_i}, ordered by $\preceq$. From the definition of Θ we deduce that U is of the form

$$(\lambda x_{\xi_{0,1}}^{\alpha_0\bullet}.\cdots.\lambda x_{\xi_{0,n_0}}^{\alpha_0\bullet}.R')\,S_{0,1}\,\ldots\,S_{0,n_0}\,S_{1,1}\,\ldots\,S_{1,n_1}\,\ldots\,S_{s,1}\,\ldots\,S_{s,n_s}\,,$$

where $R' \in \Theta(R, (\chi_1, \ldots, \chi_s), [(x^{\alpha_0}, \chi_0)]\restriction_R)$, and $\chi_i \in \mathit{Val}(S_i, [\,])$ for $i \in \{0, \ldots, s\}$, and $S_{i,j} \in \Theta(Q, \xi_{i,j}, [\,])$ for $i \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n_i\}$. As V we take

$$R'[S_{0,1}/x_{\xi_{0,1}}^{\alpha_0\bullet}] \ldots [S_{0,n_0}/x_{\xi_{0,n_0}}^{\alpha_0\bullet}]\,S_{1,1}\,\ldots\,S_{1,n_1}\,\ldots\,S_{s,1}\,\ldots\,S_{s,n_s}\,.$$

Clearly $U \xrightarrow{h}_\beta^{n_0} V$, and $n_0 \geq 1$. From Lemma B.5 we obtain that $R'[S_{0,1}/x_{\xi_{0,1}}^{\alpha_0\bullet}] \ldots [S_{0,n_0}/x_{\xi_{0,n_0}}^{\alpha_0\bullet}] \in \Theta(R[S_0/x^{\alpha_0}], (\chi_1, \ldots, \chi_s), [\,])$. Using again the definition of Θ we conclude that $V \in \Theta(Q, (), [\,])$. □

Lemma B.7. *Let $P \in \mathbf{\Lambda}_m$ be a closed lambda-term of sort o. If $U \in \Theta(P, (), [\,])$ then $BT(U)$ is a $\Xi^m_{\mathcal{M}}$-reflection of $BT(P)$.*

Proof: Coinduction on the structure of $BT(P)$. Suppose first that $P \xrightarrow{h}{}^*_\beta a\langle Q_1, \ldots, Q_r\rangle$. We have that $BT(P) = a\langle BT(Q_1), \ldots, BT(Q_r)\rangle$. Because $U \in \Theta(P, (), [])$, using Lemma B.6 consecutively for every head beta-reduction in a sequence witnessing that $P \xrightarrow{h}{}^*_\beta a\langle Q_1, \ldots, Q_r\rangle$, we obtain a lambda-term $V \in \Theta(a\langle Q_1, \ldots, Q_r\rangle, (), [])$ such that $U \xrightarrow{h}{}^*_\beta V$. From the definition of Θ it follows that V is of the form $(a, \chi)\langle R_1, \ldots, R_r\rangle$, where $R_i \in \Theta(Q_i, (), [])$ for $i \in \{1, \ldots, r\}$, and $\chi \in Val(a\langle Q_1, \ldots, Q_r\rangle, [])$. By the assumption of coinduction we have that $BT(R_i)$ is a $\Xi^m_{\mathcal{M}}$-reflection of $BT(K_i)$. Moreover, by the definition of Val, there is a closed lambda-term Q of complexity at most m, such that $\chi = [\![Q]\!]_{\mathcal{M}}$ and $Q \to^*_\beta a\langle Q_1, \ldots, Q_r\rangle$. Then $BT(Q) = BT(a\langle Q_1, \ldots, Q_r\rangle) = BT(P)$ (because the Böhm tree does not change during beta-reductions), so $(BT(P), (a, \chi)) \in \Xi^m_{\mathcal{M}}$. It follows that $BT(U)$ is a $\Xi^m_{\mathcal{M}}$-reflection of $BT(P)$.

It remains to consider the situation when there is no sequence of head beta-reductions from P to a lambda-term starting with a node constructor. Then we have one of two cases:

- there is an infinite sequence of head beta-reductions starting in P, or
- $P \xrightarrow{h}{}^*_\beta Q$ for a lambda-term Q from which no head beta-reduction can be performed, and which does not start with a node constructor.

Moreover, $BT(P) = \mathsf{nd}\langle\rangle$ and $(\mathsf{nd}\langle\rangle, \mathsf{nd}) \in \Xi^m_{\mathcal{M}}$, so it is enough to prove that $BT(U) = \mathsf{nd}\langle\rangle$. In the former case we can apply Lemma B.6 to every reduction in the infinite sequence of head beta-reductions starting from P, and we obtain an infinite sequence of head beta-reductions starting from U. This implies that from U we cannot reach a lambda-term starting with a node constructor, and thus indeed $BT(U) = \mathsf{nd}\langle\rangle$. In the latter case, we observe that the only possible form of a closed lambda-term Q of sort o from which no head beta-reduction can be performed and which does not start with a node constructor is an infinite sequence of applications $Q = \ldots Q_3\, Q_2\, Q_1$. By applying Lemma B.6 consecutively for every head beta-reduction in a sequence witnessing $P \xrightarrow{h}{}^*_\beta Q$, we obtain a lambda-term $V \in \Theta(Q, (), [])$ such that $U \xrightarrow{h}{}^*_\beta V$. From the definition of Θ it follows that V is also an infinite sequence of applications, which implies that $BT(U) = \mathsf{nd}\langle\rangle$, as needed. □

Having all this, we can easily finish the correctness proof. Indeed, $\Lambda(\mathcal{G}_{\mathcal{M}}) = \Lambda_{\mathcal{G}_{\mathcal{M}}}(N_{0,()}) = \Lambda_{\mathcal{G}_{\mathcal{M}}}(\theta(N_0, (), [])) \in \Theta(\Lambda_{\mathcal{G}}(N_0), (), []) = \Theta(\Lambda(\mathcal{G}), (), [])$ by Lemma B.3, and thus $BT(\Lambda(\mathcal{G}_{\mathcal{M}}))$ is a $\Xi^m_{\mathcal{M}}$-reflection of $BT(\Lambda(\mathcal{G}))$ by Lemma B.7.

B.3 Size and running time

It remains to bound the size of $\mathcal{G}_{\mathcal{M}}$, and the time needed to construct it. Recall that $\mathcal{G}$ is of order at most m. Denote

$$d = \left(\max_{\substack{\alpha \in \mathcal{S}_{\mathcal{G}} \\ ord(\alpha) \leq m-1}} |D[\alpha]| \right)^{|\mathcal{G}|}.$$

Our goal is to prove that $|\mathcal{G}_{\mathcal{M}}| \leq |\mathcal{G}| \cdot d^{|\mathcal{G}|}$. This is done in subsequent lemmata. First, we bound the size of the sets Arg^α and of the new sorts $\alpha^\bullet$.

Lemma B.8. *For every $\alpha \in \mathcal{S}_{\mathcal{G}}$ it is the case that $|Arg^\alpha| \leq d$.*

Proof: Denoting $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o$ we recall that $Arg^\alpha = D[\alpha_1] \times \cdots \times D[\alpha_s]$. Lemma 9.5 implies that $|\alpha| \leq 2 \cdot |\mathcal{G}|$ for $\alpha \in \mathcal{S}_{\mathcal{G}}$, hence $s \leq |\mathcal{G}|$. Moreover, because $\alpha \in \mathcal{S}_{\mathcal{G}}$ we have $ord(\alpha) \leq m$, and thus $\alpha_i \in \mathcal{S}_{\mathcal{G}}$ and $ord(\alpha_i) \leq m - 1$ for all $i \in \{1, \ldots, s\}$. We easily conclude recalling the definition of d. □

Lemma B.9. *For every $\alpha \in \mathcal{S}_\mathcal{G}$ it is the case that $|\alpha^\bullet| \leq |\alpha| \cdot d^{|\alpha|}$.*

Proof: The proof is by induction on the structure of α. For $\alpha = o$ also $\alpha^\bullet = o$, and the thesis is immediate. Consider the case when $\alpha = \beta \to \gamma$; then

$$\alpha^\bullet = \underbrace{\beta^\bullet \to \ldots \to \beta^\bullet}_{|Arg^\beta|} \to \gamma^\bullet \,.$$

Recall that $|\alpha| = |\beta| + 1 + |\gamma|$. Using the induction assumption and Lemma B.8 we obtain that

$$|\alpha^\bullet| = (|\beta^\bullet| + 1) \cdot |Arg^\beta| + |\gamma^\bullet| \leq |\beta| \cdot d^{|\beta|} \cdot d + d + |\gamma| \cdot d^{|\gamma|} \leq (|\beta| + 1 + |\gamma|) \cdot d^{|\alpha|} = |\alpha| \cdot d^{|\alpha|} \,.$$

□

Next, we bound the size of lambda-terms.

Lemma B.10. *When R^α is a subterm of $\mathcal{R}(N)$ for some nonterminal N, and $\zeta \in Arg^\alpha$, and $\nu \in Free(R)$, it is the case that $|\theta(R, \zeta, \nu)| \leq |R| \cdot d^{|R|}$.*

Proof: The proof is by induction on the structure of R. We have four cases, depending on the shape of R.

Suppose first that $R = a\langle P_1, \ldots, P_r\rangle$. Then $|R| = 1 + |P_1| + \cdots + |P_r|$, and, by the induction assumption, $|\theta(P_i, \zeta, \nu{\restriction_{P_i}})| \leq |P_i| \cdot d^{|P_i|} \leq |P_i| \cdot d^{|R|}$ for all $i \in \{1, \ldots, r\}$. We thus have

$$\begin{aligned} |\theta(R, \zeta, \nu)| &= |(a, [\![\Lambda_\mathcal{G}(R)]\!]^\nu_\mathcal{M})\langle \theta(P_1, \zeta, \nu{\restriction_{P_1}}), \ldots, \theta(P_r, \zeta, \nu{\restriction_{P_r}})\rangle| \\ &= 1 + |\theta(P_1, \zeta, \nu{\restriction_{P_1}})| + \cdots + |\theta(P_r, \zeta, \nu{\restriction_{P_r}})| \\ &\leq d^{|R|} + |P_1| \cdot d^{|R|} + \cdots + |P_r| \cdot d^{|R|} = |R| \cdot d^{|R|} \,. \end{aligned}$$

In the case of $R = x^\alpha$ we simply have that $|\theta(R, \zeta, \nu)| = |x^{\alpha^\bullet}_\zeta| = 1 \leq |R| \cdot d^{|R|}$.

Suppose now that $R = P^{\beta \to \alpha} Q^\beta$. Let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$. Clearly $\beta \in \mathcal{S}_\mathcal{G}$, so $n = |Arg^\beta| \leq d$ by Lemma B.8. Using the induction assumption we conclude that

$$\begin{aligned} |\theta(R, \zeta, \nu)| &= |\theta(P, [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P})\, \theta(Q, \xi_1, \nu{\restriction_Q}) \ \ldots \ \theta(Q, \xi_n, \nu{\restriction_Q})| \\ &= |\theta(P, [\![\Lambda_\mathcal{G}(Q)]\!]^{\nu{\restriction_Q}}_\mathcal{M} :: \zeta, \nu{\restriction_P})| + 1 + |\theta(Q, \xi_1, \nu{\restriction_Q})| + \cdots + 1 + |\theta(Q, \xi_n, \nu{\restriction_Q})| \\ &\leq |P| \cdot d^{|P|} + d \cdot (1 + |Q| \cdot d^{|Q|}) \leq (|P| + 1 + |Q|) \cdot d^{|R|} = |R| \cdot d^{|R|} \,. \end{aligned}$$

Finally, suppose that $R = \lambda x^\beta . P^\gamma$. Let $\zeta = \chi :: \zeta'$, and let $\xi_1, \ldots, \xi_n$ be all the elements of Arg^β, ordered by $\preceq$. Again $n \leq d$ by Lemma B.8. Using the induction assumption and Lemma B.9 we conclude that

$$\begin{aligned} |\theta(R, \zeta, \nu)| &= |\lambda x^{\beta^\bullet}_{\xi_1} . \cdots . \lambda x^{\beta^\bullet}_{\xi_n} . \theta(P, \zeta', \nu :: (x^\beta, \chi){\restriction_P})| \\ &= n \cdot (|\beta^\bullet| + 1) + |\theta(P, \zeta', \nu :: (x^\beta, \chi){\restriction_P})| \\ &\leq d \cdot (|\beta| \cdot d^{|\beta|} + 1) + |P| \cdot d^{|P|} \leq (|\beta| + 1 + |P|) \cdot d^{|R|} = |R| \cdot d^{|R|} \,. \end{aligned}$$

□

Finally, recall that

$$|\mathcal{G}| = \sum_{N^\alpha \in \mathcal{N}} (|\alpha| + |\mathcal{R}(N^\alpha)|) \qquad \text{and} \qquad |\mathcal{G}_\mathcal{M}| = \sum_{N^\alpha \in \mathcal{N}} \sum_{\zeta \in Arg^\alpha} (|\alpha^\bullet| + |\mathcal{R}_\mathcal{M}(N^{\alpha^\bullet}_\zeta)|) \,.$$

For every nonterminal $N^\alpha \in \mathcal{N}$ and for every $\zeta \in Arg^\alpha$, recalling that $\mathcal{R}_\mathcal{M}(N_\zeta^{\alpha^\bullet}) = \theta(\mathcal{R}(N^\alpha), \zeta, [\,])$, from Lemma B.10 we obtain that $|\mathcal{R}_\mathcal{M}(N_\zeta^{\alpha^\bullet})| \leq |\mathcal{R}(N^\alpha)| \cdot d^{|\mathcal{R}(N^\alpha)|}$. Using also Lemmata B.8 and B.9, for every $N^\alpha \in \mathcal{N}$ we have that

$$\sum_{\zeta \in Arg^\alpha} (|\alpha^\bullet| + |\mathcal{R}_\mathcal{M}(N_\zeta^{\alpha^\bullet})|) \leq d \cdot (|\alpha| \cdot d^{|\alpha|} + |\mathcal{R}(N^\alpha)| \cdot d^{|\mathcal{R}(N^\alpha)|}) \leq (|\alpha| + |\mathcal{R}(N^\alpha)|) \cdot d^{|\mathcal{G}|}\,.$$

In consequence, $|\mathcal{G}_\mathcal{M}| \leq \sum_{N^\alpha \in \mathcal{N}} (|\alpha| + |\mathcal{R}(N^\alpha)|) \cdot d^{|\mathcal{G}|} = |\mathcal{G}| \cdot d^{|\mathcal{G}|}$, as we wanted.

We notice that the transformation is defined in a straightforward way, and thus the running time is essentially proportional to the size of the resulting scheme $\mathcal{G}_\mathcal{M}$. However, in the case of a node constructor and in the case of an application, we need to compute $[\![\Lambda_\mathcal{G}(R)]\!]'_\mathcal{M}$ for some subterms R appearing in $\mathcal{G}$. To this end, it is enough to use Operations (A), (B), (C) listed in the statement of Lemma 10.3 Moreover, in the case of an application, in the case of a lambda-binder, and while listing all nonterminals we need to enumerate all elements of Arg^β for appropriate sorts $\beta \in \mathcal{S}_\mathcal{G}$. When $\beta = \alpha_1 \to \ldots \to \alpha_s \to o$, by definition $Arg^\beta = D[\alpha_1] \times \cdots \times D[\alpha_s]$, thus we actually need to enumerate all elements of $D[\alpha_i]$ for all $i \in \{1, \ldots, s\}$, where $ord(\alpha_i) \leq ord(\beta) - 1 \leq m - 1$ (Operation (D) in the statement of the lemma).