Complexity of the Diagonal Problem for Recursion Schemes

Paweł Parys

University of Warsaw, Poland

Abstract

We consider nondeterministic higher-order recursion schemes as recognizers of languages of finite words or finite trees. We establish the complexity of the diagonal problem for schemes: given a set of letters A and a scheme $\mathcal{G}$, is it 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. We prove that this problem is $(m-1)$-EXPTIME-complete for word-recognizing schemes of order m, and m-EXPTIME-complete for tree-recognizing schemes of order m.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases diagonal problem, higher-order recursion schemes, intersection types, downward closure

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

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 study the complexity of the diagonal problem 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 [27] and to higher-order OI grammars [8, 20]. Collapsible pushdown systems [11] and ordered tree-pushdown systems [5] are other equivalent formalisms. Schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [3].

The goal of this paper is to establish the complexity of the diagonal problem for higher-order schemes. By a recent result by Clemente et al. [6] we know that this problem 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). In the current work, we tighten the upper bound: we prove that the diagonal problem for word-recognizing (tree-recognizing) schemes of order m is $(m-1)$-EXPTIME-complete (m-EXPTIME-complete, respectively).

Let us recall from [6] that the decidability result for the diagonal problem entailed other decidability results for recursion schemes, concerning in particular computability of the downward closure of recognized languages [28], and the problem of separability by piecewise testable languages [7]. Although our complexity result for the diagonal problem 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.

Our solution is based on an appropriate system of intersection types. Intersection types were intensively used in the context of schemes, for several purposes like model-checking [15, 18, 4, 26], pumping [16], transformations of schemes [17, 6], etc. Among such type systems we have to distinguish those [25, 22], 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 [22] we have developed a type system that allows to solve the diagonal problem for a special case of a single-letter alphabet.

Here, we generalize the last type system mentioned above to multiple letters. In 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 appearances of a in the generated word (tree) is approximated by the number of appearances of an appropriate flag in the type derivation. In effect, the diagonal problem reduces to checking whether there exist type derivations with arbitrarily many flags corresponding to every letter from the input set A.

Some further work was needed to carefully optimize the developed type system in order to obtain an algorithm achieving the optimal complexity.

Our paper is structured as follows. In Section 2 we introduce all necessary definitions. In Section 3 we introduce the type system, and we show how to use it for deciding the diagonal problem for word-recognizing schemes. Finally, Section 4 presents extensions of the algorithm, in particular to tree-recognizing schemes.

2 Preliminaries

Infinitary λ-calculus. The set of *sorts* (a/k/a simple types), constructed from a unique basic sort o using a binary operation $\to$, is defined as usual. 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: $\mathit{ord}(o) = 0$, and $\mathit{ord}(\alpha_1{\to}\dots{\to}\alpha_s{\to}o) = 1 + \max(\mathit{ord}(\alpha_1), \dots, \mathit{ord}(\alpha_s))$ for $s \geq 1$.

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

Let Σ be an infinite set of symbols (alphabet). 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 $\mathcal{V} = \{x^\alpha, y^\beta, z^\gamma, \dots\}$ 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 λ-calculus. *Infinitary λ-terms* (or just *λ-terms*) are defined by coinduction, according to the following rules:

- node constructor—if $a \in \Sigma^{\mathsf{nd}}$, and $P_1^o, \dots, P_r^o$ are λ-terms, then $(a\langle P_1^o, \dots, P_r^o\rangle)^o$ is a λ-term,[1]
- variable—every variable $x^\alpha \in \mathcal{V}$ is a λ-term,
- application—if $P^{\alpha\to\beta}$ and Q^α are λ-terms, then $(P^{\alpha\to\beta}\, Q^\alpha)^\beta$ is a λ-term, and
- λ-binder—if P^β is a λ-term and x^α is a variable, then $(\lambda x^\alpha.P^\beta)^{\alpha\to\beta}$ is a λ-term;

[1] Our node constructor differs from the standard definition in two aspects. First, one usually assumes that symbols are ranked, i.e., that the number r is determined by the choice of a. Second, typically a symbol a is considered itself as a λ-term of sort $\underbrace{o{\to}\dots{\to}o}_{r}{\to}o$, which after applying $P_1^o, \dots, P_r^o$ as arguments is equivalent to our $(a\langle P_1^o, \dots, P_r^o\rangle)^o$. These are, though, only superficial differences.

in the above, α, β, and $\alpha{\rightarrow}\beta$ are homogeneous sorts. We naturally identify λ-terms differing only in names of bound variables. We often omit the sort annotations of λ-terms, but we keep in mind that every λ-term (and every variable) has a particular sort. *Free variables* of a λ-term are defined as usual. A λ-term P is *closed* if it has no free variables.

For a λ-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 λ-terms that have finite complexity. We also define the order of a β-reduction as the order of the involved variable. More precisely, for a number $k \in \mathbb{N}$, we say that there is a β-reduction of order k from a λ-term P to a λ-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 λ-term that is built using only node constructors, i.e., not using variables, applications, nor λ-binders. A tree is Γ-*labeled* if only symbols 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 λ-terms. We write $P \to_{\mathsf{nd}} Q$ if Q is obtained from P by choosing some appearance of the nd symbol surrounded only by symbols 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 $\mathsf{nd}\langle P_1, \dots, P_r\rangle \to_{\mathsf{nd}} P_i$ for $i \in \{1, \dots, r\}$, and if $a \in \Sigma$, and $P_i \to_{\mathsf{nd}} P'_i$ for some $i \in \{1, \dots, r\}$, and $P_j = P'_j$ for all $j \in \{1, \dots, r\} \setminus \{i\}$, then $a\langle P_1, \dots, P_r\rangle \to_{\mathsf{nd}} a\langle P'_1, \dots, P'_r\rangle$. For a relation $\square$, by $\square^*$ we denote the reflexive transitive closure of $\square$. For a λ-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 λ-terms of sort o. For such a term P, its *Böhm tree* $BT(P)$ is constructed by coinduction, as follows: if there is a sequence of β-reductions from P to a λ-term of the form $a\langle P_1, \dots, P_r\rangle$ (where $a \in \Sigma^{\mathsf{nd}}$), then $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$; otherwise $BT(P) = \mathsf{nd}\langle\rangle$.

Higher-Order Recursion Schemes. We use a very loose definition of schemes. A *higher-order recursion scheme* (or just a *scheme*) is a triple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N^o_0)$, where $\mathcal{N} \subseteq \mathcal{V}$ is a finite set of nonterminals, $\mathcal{R}$ is a function that maps every nonterminal $N \in \mathcal{N}$ to a finite λ-term whose all free variables are contained in $\mathcal{N}$ and whose sort equals the sort of N, and $N^o_0 \in \mathcal{N}$ 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. The order of the scheme is defined as the maximum of complexities of $\mathcal{R}(N)$ over all its nonterminals N.

The infinitary λ-term *generated by* a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N^o_0)$, denoted $\Lambda(\mathcal{G})$, is defined as the limit of the following process starting from N^o_0: take any nonterminal N appearing in the current term, and replace it by $\mathcal{R}(N)$. Observe that in the limit we obtain a closed λ-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 λ-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, a folklore 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 (see also Appendix C.4). We make the homogeneity assumption for technical convenience.

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 λ-term P is *word-recognizing* if for every its subterm of the form $a\langle P_1,\dots,P_r\rangle$ with $a\in\Sigma$ it holds $r\leq 1$; a scheme $\mathcal{G}$ is *word-recognizing* if $\Lambda(\mathcal{G})$ is word-recognizing. In this case, all elements of $\mathcal{L}(BT(P))$ or $\mathcal{L}(\mathcal{G})$, respectively, are words.

▶ **Example 1.** Consider the higher-order recursion scheme $\mathcal{G}_1$ with two nonterminals, M^o (taken as the starting nonterminal) and $\mathsf{N}^{(o\to o)\to o}$, and with rules

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

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 λ-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 $BT(\Lambda(\mathcal{G}_1))=\mathsf{nd}\langle T_{2^0},\mathsf{nd}\langle T_{2^1},\mathsf{nd}\langle T_{2^2},\dots\rangle\rangle\rangle$, 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 first 2^i letters are chosen from $\{\mathsf{a},\mathsf{b}\}$ arbitrarily, and the last letter is c. In the following examples we will continue to consider this scheme, together with the set $A=\{\mathsf{a},\mathsf{b}\}$.

3 Type System for the Diagonal Problem

In this section we introduce a type system that allows to solve the diagonal problem for schemes.

▶ **Definition 1.** For a set of trees L and a set of symbols A, the predicate $\mathit{Diag}_A(L)$ holds if for every $n\in\mathbb{N}$ there is some $T\in L$ with at least n occurrences of every symbol from A. The *diagonal problem* for tree-recognizing order-m schemes is to decide whether $\mathit{Diag}_A(\mathcal{L}(\mathcal{G}))$ holds, given a scheme $\mathcal{G}$ of order at most m and a set A. The diagonal problem for *word-recognizing* order-m schemes is as the above, but with the restriction that $\mathcal{G}$ is word-recognizing.

▶ **Theorem 2.** *For $m\geq 1$, the diagonal problem for word-recognizing order-$(m+1)$ schemes is m-EXPTIME-complete. For $m\in\{-1,0\}$ it is NP-complete.*

Throughout the rest of this section we solve the diagonal problem for word-recognizing schemes, and thus all schemes considered here are assumed to be word-recognizing. Moreover, we fix a set of symbols A, for which we want to solve the diagonal problem.

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, beside of their order, are also identified by a symbol from A; thus we have (k,a)-flags for $k\in\mathbb{N}$ and $a\in A$. While deriving a type for a λ-term of complexity at most $m+1$, we use markers of order from the range $0,\dots,m$, and flags of order from the range $1,\dots,m+1$.

Let P_{m+1} be a λ-term 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 the information that T has many appearances of every symbol from 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 β-reductions from P_{m+1}. Moreover, thanks to the fact that all sorts are homogeneous, the β-reductions can be rearranged so that those of higher order are performed first. Namely, we can find λ-terms $P_0,\dots,P_m$ such that

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

Some prefix of P_0 can be seen as a tree, in which we can find a path on which there are all symbols of T, and some additional nd symbols. Let us place an order-0 marker in the leaf ending this path. Additionally, for every symbol $a \in A$, we place $(1, a)$-flags in all a-labeled nodes of the considered path.

Next, we proceed back to P_1. The leaf constructor of P_0 containing our order-0 marker was created out of some particular appearance of such constructor in P_1; let us put there as well the order-0 marker. Similarly, we find node constructors in P_1 out of which in P_0 we obtain node constructors with flags, and we transfer the flags back to P_1. 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 β-reductions between P_1 and P_0 are of order 0. This means that in every such β-reduction we take a whole subtree (i.e., a λ-term of sort o) of P_1, and we replace it somewhere, possibly replicating it. But since all flags lie in P_0 on a single path, they may lie only in at most one copy of the replicated subtree. In effect, the number of appearances of order-1 flags of every kind is the same in P_1 as in P_0.

We cannot directly repeat the same reasoning to move flags from P_1 back to P_2, since now there is a problem: 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 $|A|$ paths in P_1. Namely, for every symbol $a \in A$ 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; in effect, if the number of $(1, a)$-flags was "vary large", then also the number of $(2, a)$-flags will be "very large". To do so, we choose the marked node in a clever way: staring from the root, we always proceed to this subtree in which the number of $(1, a)$-flags is the largest.

Once for every $a \in A$ all $(2, a)$-flags lie on a single path of P_1, we can transfer them back to P_2 without changing their number. Then in P_2 we again reduce to $|A|$ paths by introducing markers of order 2, 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} actually is obtainable in the process as above (in fact, we will not be labeling nodes of P_{m+1} itself, but rather nodes of a type derivation for P_{m+1}).

Type Judgments. Recall that A is the set of symbols for which we want to solve the diagonal problem. For storing the 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 $k \in \mathbb{N}$ is called an order, and $a \in A$ is called a symbol. 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 symbol of this flag (if flags with multiple symbols were used, it is enough for us to remember just one of these symbols). Thus for $m \in \mathbb{N}$ we define $\mathcal{F}_m$ to contain sets $F \subseteq \{1, \dots, m\} \times A$ such that $(k, a), (k, b) \in F$ implies $a = b$. Such sets F 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 symbol, only an order. We thus define $\mathcal{M}_m$ to contain functions $M : \mathbb{N} \to \{0, \dots, |A|\}$ such that $M(0) \leq 1$ and $M(k) = 0$ for all $k > m$. Such functions M are called *m-bounded marker multisets.*

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 $m\in\mathbb{N}$ 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.

If $\alpha=\alpha_1{\to}\dots{\to}\alpha_s{\to}o$, the set $\mathcal{T}^\alpha$ contains types that are of the form $C_1{\to}\dots{\to}C_s{\to}o$, where $C_i\in\mathcal{TC}^{\alpha_i}$ for $i\in\{1,\dots,s\}$.

Type triples in $\mathcal{TT}^\alpha_m$ are just triples $(F,M,C_1{\to}\dots{\to}C_s{\to}o)\in\mathcal{F}_m\times\mathcal{M}_m\times\mathcal{T}^\alpha$, where $M(k)=0$ for all $(k,a)\in F$, and where $M(0)+\sum_{i=1}^{s}\mathsf{Mk}(C_i)(0)=1$ (we will define $\mathsf{Mk}(C_i)$ soon). These triples store a type, together with the 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 (multi)sets of type triples that have to be derived for an argument of a λ-term, or for a λ-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, in $\mathcal{TC}^\alpha$ we have functions $C\colon\mathcal{TT}^\alpha_{ord(\alpha)}\to\{0,\dots,|A|\}$ such that $C(\hat\tau)\leq 1$ if $\mathsf{Mk}(\hat\tau)=\mathbf{0}$. For $C\in\mathcal{TC}^\alpha$ 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 $\{\!|\hat\tau_1,\dots,\hat\tau_n|\!\}$ or $\{\!|\hat\tau_i\mid i\in\{1,\dots,n\}|\!\}$ to denote the triple container C such that $C(\hat\sigma)=|\{i\in\{1,\dots,n\}\mid\hat\tau_i=\hat\sigma\}|$ for every unbalanced type triple $\hat\sigma$, and $C(\hat\sigma)=1\Leftrightarrow\exists i\in\{1,\dots,n\}\,.\,\hat\tau_i=\hat\sigma$ 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 $\mathcal{TC}^\alpha_{ord(\alpha)}$. 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.

A *type judgment* is of the form $\Gamma\vdash_m P:\hat\tau\triangleright c$, where Γ is a type environment, $m\in\mathbb{N}$ is called the *order* of the type judgment, P is a λ-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 $A\to\mathbb{N}$ called a *flag counter.*

As usually for intersection types, the intuitive meaning of a type $C{\to}\tau$ is that a λ-term having this type can return a λ-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 in our derivation for free variables); we, however, do not include markers provided by arguments of the λ-term (i.e. coming from the triple containers C_i when $\tau = C_1 \rightarrow \ldots \rightarrow C_s \rightarrow o$). The set F contains an 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, i.e., 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 function c, called a *flag counter*, that for each symbol 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 λ-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 for all x and all unbalanced type triples $\hat{\tau}$ it holds $\Gamma(x)(\hat{\tau}) + \Gamma'(x)(\hat{\tau}) \leq |A|$ (so that $(\Gamma \sqcup \Gamma')(x)$ is indeed a valid triple container). Let us also remark that rules of the type system will guarantee that the order m of all type judgments used in a derivation will be the same.

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

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

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

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 (page 4), which says that a marker of order k can be put in a place that will be a leaf after performing all β-reductions of order at least k. Indeed, the variable x remains a leaf after performing β-reductions of orders greater than $ord(x)$, but while performing β-reductions of order $ord(x)$ this leaf will be replaced by a subterm substituted for x. Recall also that, by 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 appropriate sort α; this introduces a bound on maximal numbers that may appear in F and M.

▶ **Example 2.** Denoting $\hat{\rho}_0 = (\emptyset, \{\!|0|\!\}, o)$ we can derive:

$$\frac{}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0|\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!|0|\!\}, o) \triangleright \mathbf{0}}\ (\textsc{Var}) \qquad \frac{}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat{\rho}_0|\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!|0, 1, 1|\!\}, o) \triangleright \mathbf{0}}\ (\textsc{Var})$$

In the derivation on the right, two markers of order 1 are placed in the conclusion of the rule.

We see that 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 λ-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 the 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 the 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.** In this example we show how the (Nd) and (λ) rules can be used. 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.

$$\dfrac{\dfrac{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\rho_0|\!\}]\vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\},\{\!|0|\!\},o)\triangleright\mathbf{0}}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\rho_0|\!\}]\vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle,\mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\},\{\!|0|\!\},o)\triangleright\mathbf{0}}\,(\textsc{Nd})}{\varepsilon\vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle,\mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\},\mathbf{0},\{\!|\hat\rho_0|\!\}\to o)\triangleright\mathbf{0}}\,(\lambda)$$

$$\dfrac{\dfrac{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\rho_0|\!\}]\vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset,\{\!|0,1,1|\!\},o)\triangleright\{(\mathsf{a},1),(\mathsf{b},0)\}}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\rho_0|\!\}]\vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle,\mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset,\{\!|0,1,1|\!\},o)\triangleright\{(\mathsf{a},1),(\mathsf{b},0)\}}\,(\textsc{Nd})}{\varepsilon\vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle,\mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset,\{\!|1,1|\!\},\{\!|\hat\rho_0|\!\}\to o)\triangleright\{(\mathsf{a},1),(\mathsf{b},0)\}}\,(\lambda)$$

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 will appear 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 $k\in\{1,\dots,m+1\}$ consecutively, we decide which flags of order k should be placed in the considered node of a type derivation. 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 an information about a $(k-1,a)$-flag. By receiving this 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 $a\in A$, 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

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

where, for $k \in \{0, \dots, m+1\}$ and $a \in A$,

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

We now present rules for node constructors using symbols other than nd:

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

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

In these rules we do not claim that the set $\{(0,a)\}$ passed to Comp_m is an element of $\mathcal{F}_m$ (and in fact it is not, because the order is 0, which is forbidden for flags; we also do not necessarily have that $a \in A$). The effect of passing this set is that if $M(0) > 0$ (i.e., we are on the path to the order-0 marker) and $a \in A$, then Comp_m places a $(1,a)$-flag in the current node, and maybe also some (k,a)-flags for higher k. In the $(\textsc{Con}0)$ rule, i.e., if we are in a leaf, we are allowed to place markers of arbitrary order: the marker multiset M may be arbitrary.

▶ **Example 4.** The $(\textsc{Con}1)$ rule may be instantiated in the following ways:

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat\rho_0|\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!|0|\!\}, o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat\rho_0|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\}, \{\!|0|\!\}, o) \triangleright \mathbf{0}} \;(\textsc{Con}1)$$

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!|\hat\rho_0|\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!|0,1,1|\!\}, o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!|\hat\rho_0|\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset, \{\!|0,1,1|\!\}, o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}} \;(\textsc{Con}1)$$

In the first example, a $(1,\mathsf{a})$-flag is placed in the conclusion of the rule (because the marker multiset contains 0, the pair $(0,\mathsf{a})$ passed to the Comp_1 predicate results in the $(1,\mathsf{a})$ pair in the flag set). In the second example, $(1,\mathsf{a})$- and $(2,\mathsf{a})$-flags are placed in the conclusion of the $(\textsc{Con}1)$ rule: since 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:

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

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 order $k \leq \mathit{ord}(Q)$ are visible to P, while only flags of order $k > \mathit{ord}(Q)$ are passed to the Comp_m predicate. This can be justified if we recall the intuitions staying behind the type system (see page 4). Indeed, while considering flags and markers of order k, we should imagine the λ-term obtained from the current λ-term by performing all β-reductions of order at least k; the distribution of flags and markers of order

k in the current λ-term actually simulates their distribution in this imaginary λ-term. Thus, if $k \leq ord(Q)$, then our application will disappear in this imaginary λ-term (thanks to the homogenity assumption), and Q will be already substituted somewhere in P; for this reason we need to pass the information about flags and markers of order k from Q to P. Conversely, if $k > ord(Q)$, then in the imaginary λ-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 some flags from F' cannot disappear) is useful for proofs in the appendix.

▶ **Example 5.** Recalling that $\hat{\rho}_0 = (\emptyset, \{\!\!\{0\}\!\!\}, o)$, denote by $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ the type triples derived in Example 3: $\hat{\tau}_{\mathsf{a}} = (\{(1, \mathsf{a})\}, \mathbf{0}, \{\!\!\{\hat{\rho}_0\}\!\!\} \to o)$ and $\hat{\tau}_{\mathsf{m}} = (\emptyset, \{\!\!\{1, 1\}\!\!\}, \{\!\!\{\hat{\rho}_0\}\!\!\} \to o)$. We can derive:

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

Below the lower (@) rule the information about a $(1, \mathsf{a})$-flag (from the first premiss) meets the 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 $\hat{\rho}_m = (\emptyset, M_m^{all}, o)$, where $M_m^{all} \in \mathcal{M}_m$ is such that $M_m^{all}(0) = 1$ and $M_m^{all}(k) = |A|$ for all $k \in \{1, \ldots, m\}$. The key property of the type system is described by the following theorem.

▶ **Theorem 3.** *Let $m \in \mathbb{N}$, and let P be a closed word-recognizing λ-term of sort o and complexity at most $m + 1$. Then $Diag_A(\mathcal{L}(BT(P)))$ holds if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m P : \hat{\rho}_m \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$.*

We omit the proof of this theorem. The overall idea is to follow the intuitions described on page 4, and consider only such sequences of β-reductions in which reductions of higher orders are performed before β-reductions of lower orders. The details are tedious, but rather standard. Actually, a quite similar proof was performed in our recent work [22] concerning the single-letter case. See the appendix for more details.

▶ **Example 6.** Denote $\hat{\sigma}_R = (\emptyset, \{\!\!\{0\}\!\!\}, \{\!\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{b}}, \hat{\tau}_{\mathsf{m}}\}\!\!\} \to o)$, where $\hat{\tau}_{\mathsf{b}} = (\{(1, \mathsf{b})\}, \mathbf{0}, \{\!\!\{\hat{\rho}_0\}\!\!\} \to o)$. We can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \mathbf{0}$ by descending to the first child of the outermost $\mathsf{nd}\langle\cdot, \cdot\rangle$ in $R_1 = \lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. Then, basing on a type judgment $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \{(\mathsf{a}, c(\mathsf{a}) + 1), (\mathsf{b}, c(\mathsf{b}))\}$ using in particular the derivation fragment from Example 5, and similarly $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \{(\mathsf{a}, c(\mathsf{a})), (\mathsf{b}, c(\mathsf{b}) + 1)\}$. By composing the above derivation fragments, we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ for c that is arbitrarily large on both coordinates. Examples 2-4 contain derivations of type triples $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ for the λ-term $\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle$; similarly we can derive the type triple $\hat{\tau}_{\mathsf{b}}$. Using the (@) rule one more time, we can derive $\varepsilon \vdash_1 \Lambda(\mathcal{G}_1) : \hat{\rho}_1 \triangleright c$ for c that is arbitrarily large on both coordinates.

▶ **Example 7.** Consider the scheme $\mathcal{G}_2$ obtained from $\mathcal{G}_1$ by changing the rule $\mathcal{R}(\mathsf{N})$ to $\lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), \mathsf{N}\,(\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y})\rangle$. Then while deriving a type for $\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y}$ we can use only one type triple: either $\hat{\tau}_{\mathsf{a}}$, or $\hat{\tau}_{\mathsf{b}}$, or $\hat{\tau}_{\mathsf{m}}$, which causes that the flag counter is not increased. Thus, by adopting the derivation fragment considered in the previous example, out of $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright c$ we can only derive $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright c$, with the same flag counter (where R_2 is defined analogously to R_1). Altogether, we can derive $\varepsilon \vdash_1 \Lambda(\mathcal{G}_2) : \hat{\rho}_1 \triangleright c$ only for c with $c(\mathsf{a}) + c(\mathsf{b}) \leq 1$. This corresponds to the fact that $\mathcal{L}(\mathcal{G}_2)$ contains only words with at most one letter from $\{\mathsf{a}, \mathsf{b}\}$.

▶ **Example 8.** In the derivation from Example 6 both order-1 markers were placed in the same leaf, corresponding to the subterm x. Consider, however, 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 appearances of the a symbol) in a leaf corresponding to x, and another order-1 marker (responsible for counting appearances of the b symbol) in a leaf corresponding to $\mathsf{c}\langle\rangle$.

Effectiveness. We now justify how Theorem 2 follows from Theorem 3, i.e., how given a word-recognizing scheme $\mathcal{G}$ of order at most $m+1$ and a set of symbols A, one can check in m-EXPTIME whether $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ can be derived for flag counters c that are arbitrarily large on every coordinate. Let us say that two type judgments are equivalent if they differ only in values of the flag counter. One can see that if c is required to be large enough on every coordinate, then in the derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$, for every symbol $a \in A$, there are two equivalent type judgments lying on the same path (the path may depend on a) and such that the a-coordinate of their flag counter differs. A derivation having this property will be called *pumpable*. This name is justified, because the opposite implication also holds: if we have a pumpable type derivation, then we can repeat (as many times as we want) its fragment between all these pairs of equivalent type judgments, increasing arbitrarily all coordinates of the flag counter in the resulting type judgment. This holds thanks to the following additivity property of our type system: if out of $\Gamma \vdash_m P : \hat{\tau} \triangleright c$ we can derive $\Gamma' \vdash_m P' : \hat{\tau}' \triangleright c'$, then out of $\Gamma \vdash_m P : \hat{\tau} \triangleright d$ we can derive $\Gamma' \vdash_m P' : \hat{\tau}' \triangleright c' + d - c$.

We exploit the above equivalence, and we also observe that while deriving $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ we may only use type judgments $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ (call them *useful*) in which Q is a subterm of $\Lambda(\mathcal{G})$ and $\Gamma(x) \neq \mathbf{0}$ only for variables x that are free in Q. It is not difficult to give an algorithm that checks whether there is a pumpable derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ (for some c), and works in time polynomial in the number of equivalence classes of useful type judgments (and exponential in $|A|$).

It remains to bound the number of these equivalence classes. We first bound the number of type triples. Essentially, type triples in $\mathcal{TT}_k^\alpha$ with $\alpha = \alpha_1 \rightarrow \dots \rightarrow \alpha_s \rightarrow o$ store "sets" of type triples from $\mathcal{TT}_{ord(\alpha_i)}^{\alpha_i}$, and thus their number grows exponentially when we increase the order of α by one. There is a slight exception for α of order 1: the number of type triples in $\mathcal{TT}_k^\alpha$ for such α is polynomial, not exponential. The reason is that, for a type $(C_1 \rightarrow \dots \rightarrow C_s \rightarrow o) \in \mathcal{T}_k^\alpha$ with $ord(\alpha) = 1$, the triple containers $C_1, \dots, C_s$ can contain altogether only at most one type triple (by definition it is required that $\sum_{i=1}^s \mathsf{Mk}(C_i)(0) \leq 1$ while, on the other hand, for type triples $\hat{\sigma} \in \mathcal{TT}_0^o$ it is required that $\mathsf{Mk}(\hat{\sigma})(0) = 1$). In effect, $|\mathcal{TT}_m^\alpha|$ for $ord(\alpha) \leq m+1$ is m-fold exponential in the size of $\mathcal{G}$. It easily follows that the number of equivalence classes of useful type judgments is also m-fold exponential in the size of $\mathcal{G}$, because all variables appearing in $\Lambda(\mathcal{G})$ are of order strictly smaller than $m+1$.

A trivial reduction from the problem of emptiness of $\mathcal{L}(\mathcal{G})$ shows that our problem is indeed m-EXPTIME-hard [19]. NP-completeness for $m \in \{-1, 0\}$ is proven in Appendix J.

4 Extensions

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*, i.e., languages of the form $Y_0^*\{x_1,\varepsilon\}Y_1^*\dots\{x_n,\varepsilon\}Y_n^*$, where $x_1,\dots,x_n$ are letters, and $Y_0,\dots,Y_n$ are sets of letters. The main interest on the diagonal problem comes from the fact that this problem is closely related to computability of the downward closure of a language of words (where we aim in presenting the results 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 the diagonal problem (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 [2] may shed some new light on this subject (while pumping lemmata for deterministic schemes [13, 16] seem irrelevant here).

Instead of actually computing the downward closure, Zetzsche [29] 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 know how to do this, we can at least give a partial result.

▶ **Theorem 4.** *Let $m\geq 1$. Given a word-recognizing scheme $\mathcal{H}$ of order at most $m+1$, and an ideal I, the problem of deciding whether $I\subseteq\mathcal{L}(\mathcal{H})\!\downarrow$ is m-EXPTIME-complete.*

This is an easy consequence of Theorem 2: it is enough to appropriately combine $\mathcal{H}$ with I, and then solve the diagonal problem.

Tree-Generating Schemes. Although the main interest on the diagonal problem is for word-recognizing schemes, the problem can be also considered for tree-recognizing schemes. Let us see how our methods can be adopted to this more general case. Consider a tree $T\in\mathcal{L}(\mathcal{G})$, and a term P_0 such that $\Lambda(\mathcal{G})\to_\beta^* P_0$ and $P_0\to_{\mathsf{nd}}^* T$, i.e., that T can be found in a prefix of P_0. In the word case, we were placing order-1 flags in node constructors of T, and then we continued using the fact that they are all aligned along one path (as actually T consisted of a single path). This is no longer possible in the tree case. In order to resolve this issue, we additionally use flags of order 0, and we place them in node constructors of T (dispersed on multiple paths). Then, we choose only $|A|$ paths, by placing order-1 markers in $|A|$ leaves of P_0, and for every node labeled by a $(0,a)$-flag we place a $(1,a)$-flag in the closest ancestor that lies on a chosen path. In effect all order-1 flags are concentrated on only $|A|$ paths, and we can continue as in the word case. The described modification causes an exponential growth of the number of types, which results in the following theorem.

▶ **Theorem 5.** *For $m\geq 1$, the diagonal problem for tree-recognizing order-m schemes is m-EXPTIME-complete. For $m=0$ it is NP-complete.*

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 [21] downward closures of tree languages are regular languages of trees. We notice, however, that (unlike for words) an algorithm solving the diagonal problem 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, 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 the diagonal problem, 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.

References

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

2 Kazuyuki Asada and Naoki Kobayashi. Pumping lemma for higher-order languages. Submitted, 2017.

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

4 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

5 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh 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. doi:10.4230/LIPIcs.FSTTCS.2015.163.

6 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan 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. doi:10.1145/2933575.2934527.

7 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor 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. doi:10.1007/978-3-319-22177-9_14.

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

9 Joost Engelfriet. Iterated stack automata and complexity classes. *Inf. Comput.*, 95(1):21–75, 1991. doi:10.1016/0890-5401(91)90015-T.

10 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak 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. doi:10.1145/2837614.2837627.

11 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier 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. doi:10.1109/LICS.2008.34.

12 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James 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. doi:10.1007/978-1-4684-2001-2_9.

13 Alexander Kartzow and Paweł Parys. Strictness of the collapsible pushdown hierarchy. In Branislav Rovan, Vladimiro Sassone, and Peter 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. doi:10.1007/978-3-642-32589-2_50.

14 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

15 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 416–428. ACM, 2009. doi:10.1145/1480881.1480933.

16 Naoki 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. doi:10.1109/LICS.2013.46.

17 Naoki Kobayashi, Kazuhiro Inaba, and Takeshi Tsukada. Unsafe order-2 tree languages are context-sensitive. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2014. doi:10.1007/978-3-642-54830-7_10.

18 Naoki Kobayashi and C.-H. Luke 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. doi:10.1109/LICS.2009.29.

19 Naoki Kobayashi and C.-H. Luke 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.

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

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

22 Paweł Parys. Intersection types and counting. In Naoki 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, 2017. doi:10.4204/EPTCS.242.6.

23 Paweł 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. doi:10.1109/LICS.2012.62.

24 Paweł Parys. How many numbers can a lambda-term contain? In Michael Codish and Eijiro 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. doi:10.1007/978-3-319-07151-0_19.

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

26 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter

Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

27 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible push-down automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

28 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina 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. doi:10.1007/978-3-662-47666-6_35.

29 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.

A Related work

Our Property Is Not Regular. We remark the language of infinite trees T for which $Diag_A(\mathcal{L}(T))$ holds is not regular (due to a simple pumping argument)—the diagonal problem talks about unboundedness of some quantities. This makes the problem inaccessible for 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.

Other Type Systems. The idea of using intersection types for counting is not completely new. In [24] there is a type system that, essentially, allows to estimate the size of the β-normal form of a (finite) λ-term just by looking at (the number of some flags in) a derivation of a type for this term. A similar idea, but for higher-order pushdown automata, is present in [23], where we can estimate the number $\sharp$ symbols appearing on a particular, deterministically chosen branch of the generated tree. This previous approach also uses intersection types, where the derivations are marked with just one kind of flags, denoting "productive" places of a term (oppositely to our approach, where we have different flags for different orders, and we also have markers). The trouble with the "one-flag" approach is that it works well only in a completely deterministic setting, where looking independently at each node of the Böhm tree we know how it contributes to the result; the method stops working (or at least we do not know how to prove that it works) in our situation, where we first nondeterministically perform some guesses in the Böhm tree, and only after that we want to count something that depends on the chosen values.

Relation to [6]. Our type system and the type system from [22] are, to some extent, motivated by the algorithm of [6] solving the diagonal problem. 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 the diagonal problem 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", 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.

Relation to [22]. Let us compare our type system with the type system introduced in [22] in order to solve the diagonal problem in the case of $|A| = 1$. The first difference is that we solve the case of multiple symbols in A. This is done by replacing a single marker and a single kind of flags of every order by $|A|$ markers and $|A|$ kinds of flags, one per each symbol of A. This makes the proofs slightly more complex, but seeing [6] and [10] it was quite natural that every symbol from A requires separate markers and flags.

Conceptually, it was more difficult to establish the optimal complexity. In [22] 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 proposed a very naive algorithm for checking whether there exists a pumpable type derivation: list all type derivations of height smaller than some number, and search among them for a pumpable derivation. This algorithm is doubly exponential in the number of type triples, but it is not difficult to 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 symbol, like flags), and by storing the information only about one, nondeterministically chosen, flag of every order in flag sets, instead of the information about all kinds of flags seen so far. Finally, one exponentiation is saved by observing that in the case of word-recognizing schemes, the number of order-1 types can be also made polynomial. This is the case because in the word case there is a unique leaf in which an order-0 marker may be placed.

B Additional Definitions

In this section we introduce a few new definitions, which will be used throughout the appendix. Having two functions with values in natural numbers, $f, g\colon X \to \mathbb{N}$, we write $f \leq g$ when $f(x) \leq g(x)$ for every $x \in X$; this relation will be used to compare, e.g., flag counters. For two type environments, Γ and Γ', we say that $\Gamma \sqsubseteq \Gamma'$ if $\Gamma(x) \sqsubseteq \Gamma'(x)$ for every variable x.

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 $|\alpha{\to}\beta| = 1 + |\alpha| + |\beta|$. The size of a finite λ-term P, denoted $|P|$, is also defined by induction on its structure, as follows:

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

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 to the size of a scheme we include sizes of sorts of all bound variables and all nonterminals. Although in "reasonable cases" λ-terms using large sorts are large anyway, this is sometimes important. For example, to the size of the λ-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 λ-term would be small even for very large α. Similarly, in $\mathcal{G}$ we can have a nonterminal N of sort $\alpha{\to}o$, with 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 to the size of $\mathcal{G}$.

C Our Definition of Schemes

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

C.1 Symbols Are Unranked

In the context of higher-order recursion schemes one usually considers alphabets that are ranked. This means that every symbol $a \in \Sigma$ has assigned a number $\mathit{rank}(a)$, so that every a-labeled node has $\mathit{rank}(a)$ children. Since our definition is less restrictive, our algorithm carries over to the situation of a ranked alphabet. On the other hand, reductions of our hardness proofs (Appendix J.4) can be easily adopted to produce schemes using a ranked alphabet.

C.2 Node Constructors

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

C.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 $N^{\alpha_1\to\dots\to\alpha_s\to o}\, x_1^{\alpha_1}\, \dots\, x_s^{\alpha_s} \to P^o$, where $N \in \mathcal{N}$ is a nonterminal, and P is a finite *applicative* term whose all free variables are contained $\mathcal{N} \cup \{x_1^{\alpha_1}, \dots, x_s^{\alpha_s}\}$, and where the nd symbol is not used. By an applicative term we understand a λ-term which does not contain λ-binders. Having a scheme $\mathcal{G}_{cl}$, we define $\to_{\mathcal{G}_{cl}}$ as the smallest relation such that:

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

The language generated 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 C.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 $(N\, x_1\, \dots\, x_s \to P_1), \dots, (N\, x_1\, \dots\, x_s \to P_m)$ of $\mathcal{G}_{cl}$ concerning this nonterminal, and we take $\mathcal{R}(N) = \lambda x_1.\cdots.\lambda x_s.\mathsf{nd}\langle P_1, \dots, P_m\rangle$. 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 will now show that $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{cl}(\mathcal{G}_{cl})$.

To this end, for a λ-term P let $\Lambda_{\mathcal{G}}(P)$ be the λ-term obtained as a limit of applying recursively the following operation to P: take an appearance of some nonterminal N, and replace it by $\mathcal{R}(\mathcal{N})$. In particular $\Lambda_{\mathcal{G}}(N_0) = \Lambda(\mathcal{G})$. Moreover, denote by $\xrightarrow{h}_\beta$ the head β-reduction, i.e., $P \xrightarrow{h}_\beta Q$ if $P = (\lambda x.R)\, S\, S_1\, \dots\, S_s$ and $Q = R[S/x]\, S_1\, \dots\, S_s$. It is a well-known fact that while generating the Böhm tree of a λ-term it is enough to consider outermost β-reductions only, that is, $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$ if $P \xrightarrow{h}{}^{*}_{\beta} a\langle P_1, \dots, P_r\rangle$, and $BT(P) = \mathsf{nd}\langle\rangle$ if there is no sequence of head β-reductions to a λ-term of the form $a\langle P_1, \dots, P_r\rangle$.

Consider a finite Σ-labeled tree $T = a\langle T_1, \dots, T_r\rangle$, 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 $P \to^{n}_{\mathcal{G}_{cl}} T$ if and only if $BT(\Lambda_{\mathcal{G}}(P)) \to^{n}_{\mathsf{nd}} T$. This will imply 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 $BT(\Lambda_{\mathcal{G}}(N_0)) \to^{n}_{\mathsf{nd}} T$ for some $n \in \mathbb{N}$, while $\mathcal{L}_{cl}(\mathcal{G}_{cl})$ contains finite trees T such that $N_0 \to^{n}_{\mathcal{G}_{cl}} T$ 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 $P = N\,Q_1\,\dots\,Q_s$. 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 $P \to^{n}_{\mathcal{G}_{cl}} T$ if and only if $P = N\,Q_1\,\dots\,Q_s \to_{\mathcal{G}_{cl}} R_i[Q_1/x_1, \dots, Q_s/x_s] \to^{n-1}_{\mathcal{G}_{cl}} T$ 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 $BT(\Lambda_{\mathcal{G}}(P)) \to^{n}_{\mathsf{nd}} T$ if and only if

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

We have $R_i[Q_1/x_1, \dots, Q_s/x_s] \to^{n-1}_{\mathcal{G}_{cl}} T$ if and only if $BT(\Lambda_{\mathcal{G}}(R_i[Q_1/x_1, \dots, Q_s/x_s])) \to^{n-1}_{\mathsf{nd}} T$ 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 $P \to^{n}_{\mathcal{G}_{cl}} T$ if and only if $b = a$, $k = r$, and $P_i \to^{n_i}_{\mathcal{G}_{cl}} T_i$ for all $i \in \{1, \dots, r\}$, where $n = n_1 + \dots + n_r$. On the other hand, $BT(\Lambda_{\mathcal{G}}(P)) \to^{n}_{\mathsf{nd}} T$ if and only if $b = a$, $k = r$, and $BT(\Lambda_{\mathcal{G}}(P_i)) \to^{n_i}_{\mathsf{nd}} T_i$ for all $i \in \{1, \dots, r\}$, where $n = n_1 + \dots + n_r$. We have $P_i \to^{n_i}_{\mathcal{G}_{cl}} T_i$ if and only if $BT(\Lambda_{\mathcal{G}}(P_i)) \to^{n_i}_{\mathsf{nd}} T_i$ 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 the hardness proof so that the reductions used there will produce schemes conforming to the classic definition, which will show hardness also for such schemes.[2]

C.4 Ensuring Homogeneity

The most important non-standard assumption done in our paper is that all sorts are homogeneous. In this subsection we argue that the homogeneity assumption may be introduced without loss of generality. For purposes of this subsection, we use the name *general sort* for a sort that need not to be homogeneous. *General λ-terms* and *general schemes* are defined like λ-term and schemes, but allowing general sorts in place of homogeneous sorts; λ-terms and schemes defined previously are called homogeneous λ-terms and homogeneous schemes.

▶ **Proposition C.2.** *For every general scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^o)$ one can construct in logarithmic space a homogeneous scheme $\mathcal{H}$ that is of the same order and such that $BT(\Lambda(\mathcal{G})) = BT(\Lambda(\mathcal{H}))$.*

[2] A translation in the opposite direction is also possible, but we do not give it here, as it is more technical.

Proof. The idea is to raise the order of some sorts. For example, when we have a sort $\alpha{\to}\beta{\to}o$, and $ord(\alpha) < ord(\beta)$, then we replace the sort α by a new sort α' that is of the same order as β; this ensures the homogeneity condition $ord(\alpha') \geq ord(\beta)$ without raising the order of the whole sort $\alpha'{\to}\beta{\to}o$. The raising of the order of α is achieved by adding to it an additional parameter of an appropriate order; on the side of λ-terms this parameter will be simply ignored.

We now give more details. Let us define by induction:

$$\gamma_0 = o, \qquad\qquad \gamma_k = \gamma_{k-1}{\to}o \qquad \text{for } k \geq 1.$$

We see that $ord(\gamma_k) = k$ for all $k \in \mathbb{N}$. Next, for every sort α and every $k \geq ord(\alpha)$ we define:

$$\mathbf{R}_k(\alpha) = \begin{cases} o & \text{if } k = ord(\alpha) = 0, \\ \gamma_{k-1}{\to}\alpha & \text{otherwise.} \end{cases}$$

We see that $ord(\mathbf{R}_k(\alpha)) = k$. Basing on $\mathbf{R}_k$, we define, by induction, a transformation changing a general sort into a homogeneous one:

$$\mathbf{H}(o) = o, \qquad\qquad \mathbf{H}(\alpha{\to}\beta) = \mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha)){\to}\mathbf{H}(\beta).$$

Next, we come to transforming terms. For every $k \in \mathbb{N}$, let z^{γ_k} be a fresh variable (not appearing in the scheme that is transformed). For some fixed symbol $\mathsf{e} \in \Sigma^{\mathsf{nd}}$ we take:

$$U_0 = \mathsf{e}\langle\rangle, \qquad\qquad U_k = \lambda\mathsf{z}^{\gamma_{k-1}}.\mathsf{e}\langle\rangle \qquad \text{for } k \geq 1.$$

Clearly U_k has sort γ_k for all $k \in \mathbb{N}$. The sort of a λ-term may be changed from α to $\mathbf{R}_k(\alpha)$ by applying the following transformation, also called $\mathbf{R}_k$:

$$\mathbf{R}_k(P^\alpha) = \begin{cases} P^\alpha & \text{if } k = ord(\alpha) = 0, \\ \lambda\mathsf{z}^{\gamma_{k-1}}.P^\alpha & \text{otherwise.} \end{cases}$$

Conversely, $\mathbf{L}(P^\beta)$ lowers the sort of a λ-term from $\beta = \mathbf{R}_k(\alpha)$ back to α:

$$\mathbf{L}(P^\beta) = \begin{cases} P^\beta & \text{if } ord(\beta) = 0, \\ P^\beta\, U_{ord(\beta)-1} & \text{otherwise.} \end{cases}$$

We also define a set $\mathbf{M}(P^\alpha)$ containing all λ-terms obtained from P^α by alternately applying the operations of raising and lowering the order:

$$\mathbf{M}(P^\alpha) = \{\mathbf{L}(\mathbf{R}_{k_1}(\mathbf{L}(\mathbf{R}_{k_2}(\dots(\mathbf{L}(\mathbf{R}_{k_m}(P^\alpha)))\dots)))) \mid k_1,\dots,k_m \geq ord(\alpha)\}\,.$$

In particular we always have $P^\alpha \in \mathbf{M}(P^\alpha)$, obtained by taking $m = 0$.

A *raise environment* is a function Ω mapping some variable names to sorts. Intuitively, $\Omega(x^\alpha)$ is a new sort that the variable gets after the translation. Then, by coinduction we define a relation $\rightsquigarrow_\Omega$ between general λ-terms and corresponding homogeneous λ-terms, parameterized by a raise environment:

$$\begin{aligned}
a\langle P_1^o,\dots,P_r^o\rangle \rightsquigarrow_\Omega Q^o \quad & \text{if } Q^o \in \mathbf{M}(a\langle Q_1^o,\dots,Q_r^o\rangle), \\
& \text{and } P_i^o \rightsquigarrow_\Omega Q_i^o \text{ for all } i \in \{1,\dots,r\}, \\
x^\alpha \rightsquigarrow_\Omega Q^{\mathbf{H}(\alpha)} \quad & \text{if } Q^{\mathbf{H}(\alpha)} \in \mathbf{M}(\mathbf{L}(x^{\Omega(x^\alpha)})) \text{ for } x^\alpha \in \mathcal{V}\setminus\mathcal{N}, \\
N^\alpha \rightsquigarrow_\Omega Q^{\mathbf{H}(\alpha)} \quad & \text{if } Q^{\mathbf{H}(\alpha)} \in \mathbf{M}(N^{\mathbf{H}(\alpha)}) \text{ for } N^\alpha \in \mathcal{N}, \\
P_1^{\alpha\to\beta}\, P_2^\alpha \rightsquigarrow_\Omega Q^{\mathbf{H}(\beta)} \quad & \text{if } Q^{\mathbf{H}(\beta)} \in \mathbf{M}(Q_1^{\mathbf{H}(\alpha\to\beta)}\,\mathbf{R}_{ord(\alpha\to\beta)-1}(Q_2^{\mathbf{H}(\alpha)})), \\
& \text{and } P_1^{\alpha\to\beta} \rightsquigarrow_\Omega Q_1^{\mathbf{H}(\alpha\to\beta)}, \text{ and } P_2^\alpha \rightsquigarrow_\Omega Q_2^{\mathbf{H}(\alpha)}\,, \\
\lambda x^\alpha.P_1^\beta \rightsquigarrow_\Omega Q^{\mathbf{H}(\alpha\to\beta)} \quad & \text{if } Q^{\mathbf{H}(\alpha\to\beta)} \in \mathbf{M}(\lambda x^{\alpha'}.Q_1^{\mathbf{H}(\beta)}), \text{ and } P_1^\beta \rightsquigarrow_{\Omega[x^\alpha\mapsto\alpha']} Q_1^{\mathbf{H}(\beta)}, \\
& \text{and } \alpha' = \mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha))\,.
\end{aligned}$$

We notice that the relation $\rightsquigarrow_\emptyset$ can translate every closed λ-term P^α to a homogeneous λ-term of sort $\mathbf{H}(\alpha)$.

In other words, the translation works as follows. We first change the sort of every subterm from α to $\mathbf{H}(\alpha)$. This causes a problem on applications, since to a function of sort $\mathbf{H}(\alpha{\to}\beta) = \mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha)){\to}\mathbf{H}(\beta)$ we apply an argument of sort $\mathbf{H}(\alpha)$. We thus repair the argument by applying $\mathbf{R}_{ord(\alpha\to\beta)-1}(\cdot)$ to it. This also causes a problem on λ-binders and on variables: the new sort of a λ-binder $\lambda x^\alpha.P_1^\beta$ should be $\mathbf{H}(\alpha{\to}\beta) = \mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha)){\to}\mathbf{H}(\beta)$, so the sort of the variable should be $\mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha))$; however, while using this variable, we expect that it will have sort $\mathbf{H}(\alpha)$. We thus apply $\mathbf{L}(\cdot)$ to every place where the variable is used. The raise environment is used to pass the new sort of a variable from the place where it is bound to places where it is used. There is no problem with nonterminals: every nonterminal simply changes its sort from α to $\mathbf{H}(\alpha)$. When the above is done (in a completely deterministic way), we allow to put in every place of the whole term some spare applications of $\mathbf{R}_k(\cdot)$ followed immediately by applications of $\mathbf{L}(\cdot)$.[3] This is realized by going through the set $\mathbf{M}(\cdot)$ at every step of the definition.

As the new scheme we take $\mathcal{H} = (\mathcal{N}', \mathcal{R}', N_0^o)$, where $\mathcal{N}' = \{N^{\mathbf{H}(\alpha)} \mid N^\alpha \in \mathcal{N}\}$ and $\mathcal{R}'(N^{\mathbf{H}(\alpha)}) = Q^{\mathbf{H}(\alpha)}$ for the smallest λ-term $Q^{\mathbf{H}(\alpha)}$ such that $\mathcal{R}(N^\alpha) \rightsquigarrow_\emptyset Q^{\mathbf{H}(\alpha)}$ (the smallest such λ-term is obtained by choosing at each step the smallest element of the set $\mathbf{M}(\cdot)$). It is easy to see $\Lambda(\mathcal{G}) \rightsquigarrow_\emptyset \Lambda(\mathcal{H})$, and that $\mathcal{H}$ can be computed in logarithmic space (in particular its size is polynomial in the size of $\mathcal{G}$).

We also notice that the order of the scheme (defined as the maximum of complexities of right sides of rules) remains unchanged. This is the case because $ord(\mathbf{H}(\alpha)) = ord(\alpha)$ for every sort α. Although for every application $P_1^{\alpha\to\beta}\,P_2^\alpha$ we raise the order of the argument, it is raised only to $ord(\alpha{\to}\beta) - 1$, which is anyway smaller than the order of P_1.

It remains to prove that $BT(P^o) = BT(Q^o)$ for all closed λ-terms P^o, Q^o such that $P^o \rightsquigarrow_\emptyset Q^o$. This follows immediately (by coinduction) from the next lemma. ◀

▶ **Lemma C.3.** *Let P^o, Q^o be λ-terms such that $P^o \rightsquigarrow_\emptyset Q^o$.*

1. *If $P^o \to_\beta^* a\langle P_1^o, \dots, P_r^o\rangle$, then $Q^o \to_\beta^* a\langle Q_1^o, \dots, Q_r^o\rangle$ for some λ-terms $Q_1^o, \dots, Q_r^o$ such that $P_i^o \rightsquigarrow_\emptyset Q_i^o$ for all $i \in \{1, \dots, r\}$.*
2. *If $Q^o \to_\beta^* a\langle Q_1^o, \dots, Q_r^o\rangle$, then $P^o \to_\beta^* a\langle P_1^o, \dots, P_r^o\rangle$ for some λ-terms $P_1^o, \dots, P_r^o$ such that $P_i^o \rightsquigarrow_\emptyset Q_i^o$ for all $i \in \{1, \dots, r\}$.*

Proof. Let us concentrate on point 1. We proceed by induction on the length of some fixed sequence of β-reductions witnessing $P^o \to_\beta^* a\langle P_1^o, \dots, P_r^o\rangle$. Consider first the base case, when $P^o = a\langle P_1^o, \dots, P_r^o\rangle$. Then, by definition of $\rightsquigarrow_\emptyset$, we have that $Q^o \in \mathbf{M}(a\langle Q_1^o, \dots, Q_r^o\rangle)$ for some λ-terms $Q_1^o, \dots, Q_r^o$ such that $P_i^o \rightsquigarrow_\emptyset Q_i^o$ for all $i \in \{1, \dots, r\}$. Thus Q^o is of the form $\mathbf{L}(\mathbf{R}_{k_1}(\dots(\mathbf{L}(\mathbf{R}_{k_m}(a\langle Q_1^o, \dots, Q_r^o\rangle)))\dots))$. If $k_m = 0$, then simply $\mathbf{L}(\mathbf{R}_{k_m}(a\langle Q_1^o, \dots, Q_r^o\rangle)) = a\langle Q_1^o, \dots, Q_r^o\rangle$. Otherwise

$$\mathbf{L}(\mathbf{R}_{k_m}(a\langle Q_1^o, \dots, Q_r^o\rangle)) = (\lambda \mathsf{z}^{\gamma_{k_m-1}}.a\langle Q_1^o, \dots, Q_r^o\rangle)\,U_{k_m-1} \to_\beta a\langle Q_1^o, \dots, Q_r^o\rangle$$

(recall that $\mathsf{z}^{\gamma_{k_m-1}}$ is a fresh variable, not appearing free in Q_i^o). In the same way, we eliminate further $\mathbf{L}(\mathbf{R}_{k_i}(\cdot))$, obtaining $Q^o \to_\beta^* a\langle Q_1^o, \dots, Q_r^o\rangle$.

In the induction step consider the first β-reduction from P^o. Since the definition of $\rightsquigarrow_\Omega$ is completely structural, it is enough to concentrate on the redex involved in the β-reduction,

[3] This is not needed while defining the new scheme $\mathcal{H}$, but is necessary in the correctness proof (Lemma C.3), since such spare applications of $\mathbf{L}(\mathbf{R}_k(\cdot))$ may appear during β-reductions.

and to prove that if $(\lambda x^\alpha.R_1^\beta)\,R_2^\alpha \rightsquigarrow_\Omega S^{\mathbf{H}(\beta)}$, then $R_1^\beta[R_2^\alpha/x^\alpha] \rightsquigarrow_\Omega T^{\mathbf{H}(\beta)}$ for some λ-term $T^{\mathbf{H}(\beta)}$ such that $S^{\mathbf{H}(\beta)} \to_\beta^* T^{\mathbf{H}(\beta)}$. By definition of $\rightsquigarrow_\Omega$ we have that

$$S^{\mathbf{H}(\beta)} \in \mathbf{M}(\mathbf{L}(\mathbf{R}_{k_1}(\dots(\mathbf{L}(\mathbf{R}_{k_m}(\lambda x^{\alpha'}.S_1^{\mathbf{H}(\beta)})))\dots))\,\mathbf{R}_{ord(\alpha\to\beta)-1}(S_2^{\mathbf{H}(\alpha)}))$$

with $R_1^\beta \rightsquigarrow_{\Omega[x^\alpha\mapsto\alpha']} S_1^{\mathbf{H}(\beta)}$, and $R_2^\alpha \rightsquigarrow_\Omega S_2^{\mathbf{H}(\alpha)}$, and $\alpha' = \mathbf{R}_{ord(\alpha\to\beta)-1}(\mathbf{H}(\alpha))$. As previously, by performing some β-reductions we can eliminate the prefix of $\mathbf{L}(\mathbf{R}_{k_i}(\cdot))$, as well as the other such prefix hidden in the definition of $\mathbf{M}$, obtaining

$$S^{\mathbf{H}(\beta)} \to_\beta^* (\lambda x^{\alpha'}.S_1^{\mathbf{H}(\beta)})\,\mathbf{R}_{ord(\alpha\to\beta)-1}(S_2^{\mathbf{H}(\alpha)}) \to_\beta S_1^{\mathbf{H}(\beta)}[\mathbf{R}_{ord(\alpha\to\beta)-1}(S_2^{\mathbf{H}(\alpha)})/x^{\alpha'}]\,.$$

In order to see that $R_1^\beta[R_2^\alpha/x^\alpha] \rightsquigarrow_\Omega S_1^{\mathbf{H}(\beta)}[\mathbf{R}_{ord(\alpha\to\beta)-1}(S_2^{\mathbf{H}(\alpha)})/x^{\alpha'}]$, consider now some place where x^α is used inside R_1^β. The corresponding subterm of $S_1^{\mathbf{H}(\beta)}$ is from $\mathbf{M}(\mathbf{L}(x^{\alpha'}))$; after substituting, it is some $U^{\mathbf{H}(\alpha)} \in \mathbf{M}(\mathbf{L}(\mathbf{R}_{ord(\alpha\to\beta)-1}(S_2^{\mathbf{H}(\alpha)}))) \subseteq \mathbf{M}(S_2^{\mathbf{H}(\alpha)})$. Because $R_2^\alpha \rightsquigarrow_\Omega S_2^{\mathbf{H}(\alpha)}$, we also have $R_2^\alpha \rightsquigarrow_\Omega U^{\mathbf{H}(\alpha)}$, since the definition of $\rightsquigarrow_\Omega$ allows appending any sequence of $\mathbf{L}(\mathbf{R}_i(\cdot))$ in the front. We also have $R_2^\alpha \rightsquigarrow_{\Omega'} U^{\mathbf{H}(\alpha)}$ for appropriate Ω' needed in this place, since by definition we perform the substitution so that names of bound variables in R_1^β do not overlap with names of free variables of R_2^α (thus Ω' equals Ω on all free variables of R_2^α).

Point 2 is analogous. Every β-reduction in Q^o either has a corresponding β-reduction in P^o, or it amounts to eliminating a pair $\mathbf{L}(\mathbf{R}_i(\cdot))$ somewhere in Q^o. ◀

D Tree-Generating Schemes

In this section we define a variant of our type system that works for tree-recognizing schemes. We start by defining flag sets and marker multisets. For $m \geq -1$, an *m-bounded flag $\vartriangle$-set* is defined as a set $F \subseteq \{0,\dots,m\} \times A$ such that $(k,a),(k,b) \in F$ implies $a = b$; an *m-bounded marker $\vartriangle$-multiset* is defined as a function $M\colon \mathbb{N} \to \{0,\dots,|A|\}$ such that $M(k) = 0$ for all $k > m$. Let $\mathcal{F}_m^\vartriangle$ and $\mathcal{M}_m^\vartriangle$ contain all m-bounded flag $\vartriangle$-sets and m-bounded marker $\vartriangle$-multisets, respectively. The difference with respect to the previous definition is that flag $\vartriangle$-sets, unlike flag sets, may contain pairs $(0,a)$, and that $M(0) \leq |A|$ for marker $\vartriangle$-multisets, while $M(0) \leq 1$ for marker multisets.

Next, by mutual induction on the sort α, we define the set $\mathcal{T}^{\vartriangle\alpha}$ of *$\vartriangle$-types* of sort α, the set $\mathcal{TT}_m^{\vartriangle\alpha}$ of *m-bounded type $\vartriangle$-triples* of sort α, and the set $\mathcal{TC}^{\vartriangle\alpha}$ of *triple $\vartriangle$-containers* of sort α, and basing on that we define *type $\vartriangle$-environments* and *type $\vartriangle$-judgments*. They are defined exactly as in the word case, with the exception that:

- we allow $m \geq -1$ instead of $m \geq 0$,
- we use $\mathcal{F}_m^\vartriangle$ and $\mathcal{M}_m^\vartriangle$ instead of $\mathcal{F}_m$ and $\mathcal{M}_m$, respectively,
- $\mathcal{TT}_m^{\vartriangle\alpha}$ contains all elements $(F, M, C_1\to\dots\to C_s\to o)$ of $\mathcal{F}_m^\vartriangle \times \mathcal{M}_m^\vartriangle \times \mathcal{T}_m^{\vartriangle\alpha}$ such that $M(k) = 0$ for all $(k,a) \in F$ (we drop the requirement that $M(0) + \sum_{i=1}^s \mathsf{Mk}(C_i)(0) = 1$), and
- in type $\vartriangle$-judgments we write $\vdash_m^\vartriangle$ instead of $\vdash_m$.

We now come to the type system. Its rules (Var), (Nd), (λ), (Con0), and (@) are as previously, with the only difference that we use $\vdash_m^\vartriangle$ instead of $\vdash_m$. Instead of the (Con1) rule we have the following (Con≥1) rule, which talks about node constructors of an arbitrary positive rank:

$$\frac{\begin{array}{cc}\Gamma_i \vdash_m^\vartriangle P_i : (F_i, M_i, o) \triangleright c_i \text{ for each } i \in \{1,\dots,r\} & M = M_1 + \dots + M_r \\ (F,c) \in \mathit{Comp}_m(M; (\{(0,a)\}, \mathbf{0}), (F_1,c_1),\dots,(F_r,c_r)) \quad r \geq 1 & a \neq \mathsf{nd}\end{array}}{\Gamma_1 \sqcup \dots \sqcup \Gamma_r \vdash_m^\vartriangle a\langle P_1,\dots,P_r\rangle : (F, M, o) \triangleright c}\ (\textsc{Con}{\geq}1)$$

We notice that the previous (Con1) rule can be obtained from the (Con≥1) rule by adding the requirement that $r = 1$, and replacing $\vdash^{\triangle}_m$ by $\vdash_m$.

Denote $\hat\rho^{\triangle}_m = (\emptyset, M^{\triangle all}_m, o)$, where $M^{\triangle all}_m \in \mathcal{M}^{\triangle}_m$ is such that $M^{\triangle all}_m(k) = |A|$ for all $k \in \{0, \dots, m\}$. The following theorem is an analogue of Theorem 3 for tree-recognizing schemes.

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

We prove this theorem in Appendices F–I, together with Theorem 3.

In the sequel, we use the symbol κ to denote either $\triangle$ or ε (i.e., nothing). Thus we write $\vdash^{\kappa}_m$ in statements that are true both for $\vdash_m$ and $\vdash^{\triangle}_m$; similarly $\hat\rho^{\kappa}_m$, etc.

E Examples

Example 6 Expanded. Let us give more details concerning Example 6. Recall the type triples that were already defined:

$$\begin{aligned}
&\hat\rho_0 = (\emptyset, \{\!\!\{0\}\!\!\}, o)\,, && \hat\tau_{\mathsf a} = (\{(1,\mathsf a)\}, \mathbf{0}, \{\!\!\{\hat\rho_0\}\!\!\}{\to}o)\,, && \hat\tau_{\mathsf m} = (\emptyset, \{\!\!\{1,1\}\!\!\}, \{\!\!\{\hat\rho_0\}\!\!\}{\to}o)\,,\\
&\hat\rho_1 = (\emptyset, \{\!\!\{0,1,1\}\!\!\}, o)\,, && \hat\tau_{\mathsf b} = (\{(1,\mathsf b)\}, \mathbf{0}, \{\!\!\{\hat\rho_0\}\!\!\}{\to}o)\,, && \hat\sigma_R = (\emptyset, \{\!\!\{0\}\!\!\}, \{\!\!\{\hat\tau_{\mathsf a}, \hat\tau_{\mathsf b}, \hat\tau_{\mathsf m}\}\!\!\}{\to}o)\,.
\end{aligned}$$

The type judgment $\varepsilon \vdash_1 R_1 : \hat\sigma_R \triangleright \mathbf{0}$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf m}\}\!\!\}] \vdash_1 \mathsf f : \hat\tau_{\mathsf m} \triangleright \mathbf{0}}\,(\textsc{Var}) \qquad \dfrac{}{\varepsilon \vdash_1 \mathsf c\langle\rangle : \hat\rho_0 \triangleright \mathbf{0}}\,(\textsc{Con})}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf m}\}\!\!\}] \vdash_1 \mathsf f\,(\mathsf c\langle\rangle) : \hat\rho_1 \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf m}\}\!\!\}] \vdash_1 \mathsf{nd}\langle \mathsf f\,(\mathsf c\langle\rangle), R_1\,(\lambda \mathsf y.\mathsf f\,(\mathsf f\,\mathsf y))\rangle : \hat\rho_1 \triangleright \mathbf{0}}\,(\textsc{Nd})}{\varepsilon \vdash_1 R_1 : \hat\sigma_R \triangleright \mathbf{0}}\,(\lambda)$$

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 5 we have derived the type judgment $\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}, \hat\tau_{\mathsf m}\}\!\!\}] \vdash_1 \lambda \mathsf y.\mathsf f\,(\mathsf f\,\mathsf y) : \hat\tau_{\mathsf m} \triangleright \{(\mathsf a, 1), (\mathsf b, 0)\}$. Similarly we can derive $\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf b}, \hat\tau_{\mathsf m}\}\!\!\}] \vdash_1 \lambda \mathsf y.\mathsf f\,(\mathsf f\,\mathsf y) : \hat\tau_{\mathsf m} \triangleright \{(\mathsf a, 0), (\mathsf b, 1)\}$. We continue by deriving the type triple $\hat\tau_{\mathsf a}$ for the subterm $\lambda \mathsf y.\mathsf f\,(\mathsf f\,\mathsf y)$:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}\}\!\!\}] \vdash_1 \mathsf f : \hat\tau_{\mathsf a} \triangleright \mathbf{0}} \qquad \dfrac{\dfrac{}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}\}\!\!\}] \vdash_1 \mathsf f : \hat\tau_{\mathsf a} \triangleright \mathbf{0}} \qquad \dfrac{}{\varepsilon[\mathsf y \mapsto \{\!\!\{\hat\rho_0\}\!\!\}] \vdash_1 \mathsf y : \hat\rho_0 \triangleright \mathbf{0}}}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}\}\!\!\}, \mathsf y \mapsto \{\!\!\{\hat\rho_0\}\!\!\}] \vdash_1 \mathsf f\,\mathsf y : (\{(1,\mathsf a)\}, \{\!\!\{0\}\!\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}\}\!\!\}, \mathsf y \mapsto \{\!\!\{\hat\rho_0\}\!\!\}] \vdash_1 \mathsf f\,(\mathsf f\,\mathsf y) : (\{(1,\mathsf a)\}, \{\!\!\{0\}\!\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf f \mapsto \{\!\!\{\hat\tau_{\mathsf a}\}\!\!\}] \vdash_1 \lambda \mathsf y.\mathsf f\,(\mathsf f\,\mathsf y) : \hat\tau_{\mathsf a} \triangleright \mathbf{0}}\,(\lambda)$$

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 λ-term R_1, where we denote $c_{\mathsf{a}} = \{(\mathsf{a}, 1), (\mathsf{b}, 0)\}$:

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

In this fragment of a derivation no flag nor counter is placed. In particular, there is no order-2 flag in conclusion of the (@) rule, although its second 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 the 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 $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c + c_{\mathsf{b}}$, where $c_{\mathsf{b}} = \{(\mathsf{a}, 0), (\mathsf{b}, 1)\}$. By composing these two derivation fragments, we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ for every c. Finally, we apply the argument $S = \lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle$, and we derive for $\Lambda(\mathcal{G}_1)$ the type triple $\hat{\rho}_2$, appearing in Theorem 3.

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

Recall that from Examples 2-4 we already know how to derive the three premisses concerning S. 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 $\varepsilon \vdash_1 S : \hat{\tau}_{\mathsf{m}} \triangleright c_{\mathsf{b}}$, obtaining flag counter $c + c_{\mathsf{b}}$ at the end. We can notice that there is a correspondence between a derivation with flag counter $c + c_{\mathsf{a}}$ and some tree in $\mathcal{L}(P)$ having $2^{c(\mathsf{a})+c(\mathsf{b})}$ nodes. We remark that in every of the above derivations only three flags of order 1 are present (two $(1, \mathsf{a})$-flags and one $(1, \mathsf{b})$-flag), in the three nodes using the (Con1) rule.

Example 7 Expanded. The λ-term R_2 is obtained from R_1 by replacing all appearances of the subterm $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$ with $\mathsf{f}\,\mathsf{y}$. 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 $R_2 = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda \mathsf{y}.\mathsf{f}\,\mathsf{y})\rangle$. 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 $\hat{\tau}_{\mathsf{m}}$:

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

In the subterm $\mathsf{f}\,\mathsf{y}$ we have only one appearance of f, so we cannot use simultaneously both $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ (as we did for $\mathsf{f}\,(\mathsf{f}\,\mathsf{y})$); in effect 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 effect, the type triple $\hat{\rho}_1$ will be derived for $\Lambda(\mathcal{G}_2)$ with flag counter c_{a} (or c_{b} if one prefers).

Example 7 Modified. Consider now the λ-term $P_2 = (\lambda \mathsf{g}.\Lambda(\mathcal{G}_2))\,(\lambda \mathsf{z}.P_\mathsf{a})$, where P_a is the unique λ-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 λ-term that cannot be described by any scheme). We see that P_2 β-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:

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

We notice that the last two lines can be repeated arbitrarily many times. Then, in the conclusion of every $(\textsc{Con1})$ 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:

$$\dfrac{\dfrac{\varepsilon\vdash_1 \Lambda(\mathcal{G}_2):\hat\rho_1\triangleright c_\mathsf{a}}{\varepsilon\vdash_1 \lambda\mathsf{g}.\Lambda(\mathcal{G}_2):(\emptyset,\{\!|0,1,1|\!\},\{\!|\hat\tau_\mathsf{a}|\!\}\to o)\triangleright c_\mathsf{a}}\,(\lambda)\qquad \dfrac{\varepsilon[\mathsf{z}\mapsto\{\!|\hat\rho_0|\!\}]\vdash_1 P_\mathsf{a}:(\{(1,\mathsf{a})\},\{\!|0|\!\},o)\triangleright\mathbf{0}}{\varepsilon\vdash_1 \lambda\mathsf{z}.P_\mathsf{a}:\hat\tau_\mathsf{a}\triangleright\mathbf{0}}\,(\lambda)}{\varepsilon\vdash_1 P_2:\hat\rho_1\triangleright c_\mathsf{a}}\,(@)$$

Because $\hat\tau_\mathsf{a}$ is balanced, it can be discarded in the (λ) rule, and need not 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 8 Expanded. Since this time we want to place only one order-1 marker in a leaf corresponding to x, we consider the type triple $\hat\tau'_\mathsf{m} = (\emptyset,\{\!|1|\!\},\{\!|\hat\rho_0|\!\}\to o)$. We can derive $\hat\tau'_\mathsf{m}$ and $\hat\tau_\mathsf{a} = (\{(1,\mathsf{a})\},\mathbf{0},\{\!|\hat\rho_0|\!\}\to o)$ for $\lambda\mathsf{x}.\mathsf{a}\langle\mathsf{x}\rangle$:

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

Denote by P_b the λ-term corresponding to M_b, i.e., $P_\mathsf{b} = \mathsf{nd}\langle\mathsf{c}\langle\rangle,\mathsf{b}\langle P_\mathsf{b}\rangle\rangle$. We can derive:

$$\dfrac{\dfrac{}{\varepsilon\vdash_1 \mathsf{c}\langle\rangle:(\emptyset,\{\!|0,1|\!\},o)\triangleright\mathbf{0}}}{\varepsilon\vdash_1 P_\mathsf{b}:(\emptyset,\{\!|0,1|\!\},o)\triangleright\mathbf{0}}\qquad\qquad \dfrac{\dfrac{\varepsilon\vdash_1 P_\mathsf{b}:(\emptyset,\{\!|0,1|\!\},o)\triangleright\{(\mathsf{a},0),(\mathsf{b},k)\}}{\varepsilon\vdash_1 \mathsf{b}\langle P_\mathsf{b}\rangle:(\emptyset,\{\!|0,1|\!\},o)\triangleright\{(\mathsf{a},0),(\mathsf{b},k+1)\}}}{\varepsilon\vdash_1 P_\mathsf{b}:(\emptyset,\{\!|0,1|\!\},o)\triangleright\{(\mathsf{a},0),(\mathsf{b},k+1)\}}$$

Starting with the derivation fragment on the left, and then appending the fragment on the right an appropriate number of times, we can derive $\varepsilon\vdash_1 P_\mathsf{b}:(\emptyset,\{\!|0,1|\!\},o)\triangleright\{(\mathsf{a},0),(\mathsf{b},k)\}$ for every $k\in\mathbb{N}$.

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 6. Let us take $\hat\sigma'_R = (\emptyset,\{\!|0,1|\!\},\{\!|\hat\tau_\mathsf{a},\hat\tau'_\mathsf{m}|\!\}\to o)$. Consider the λ-term R_3 corresponding to N, namely the unique λ-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:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau'_\mathsf{m}|\!\}]\vdash_1 \mathsf{f}:\hat\tau'_\mathsf{m}\triangleright\mathbf{0}}\,(\textsc{Var})\qquad \varepsilon\vdash_1 P_\mathsf{b}:(\emptyset,\{\!|0,1|\!\},o)\triangleright c}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau'_\mathsf{m}|\!\}]\vdash_1 \mathsf{f}\,P_\mathsf{b}:\hat\rho_1\triangleright c}\,(@)}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau'_\mathsf{m}|\!\}]\vdash_1 \mathsf{nd}\langle\mathsf{f}\,P_\mathsf{b},R_3\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle:\hat\rho_1\triangleright c}\,(\textsc{Nd})}{\varepsilon\vdash_1 R_3:\hat\sigma'_R\triangleright c}\,(\lambda)$$

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

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

Notice that, in the last two derivation fragments, the final (λ) rule removes one order-1 marker from the marker multiset of $\hat{\rho}_1$, so that the marker multiset of $\hat{\sigma}'_R$ contains one order-1 marker. This is because $\hat{\tau}'_\mathsf{m}$ provides one order-1 marker. In Example 6 $\hat{\tau}_\mathsf{m}$ provided two order-1 markers, and in effect $\hat{\sigma}_R$ 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. We end the derivation as in Example 6:

$$\dfrac{\varepsilon \vdash_1 R_3 : \hat{\sigma}'_R \triangleright c \qquad \varepsilon \vdash_1 \lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle : \hat{\tau}_\mathsf{a} \triangleright 0 \qquad \varepsilon \vdash_1 \lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle : \hat{\tau}'_\mathsf{m} \triangleright c_\mathsf{a}}{\varepsilon \vdash_1 \Lambda(\mathcal{G}_3) : \hat{\rho}_1 \triangleright c + c_\mathsf{a}}\,(@)$$

A Tree-Generating Scheme. In the final example we consider a tree-recognizing scheme $\mathcal{G}_4$ with the following rules:

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

In $\mathcal{L}(\mathcal{G}_4)$ we have full binary trees of height k for every $k \geq 1$, where internal nodes are labeled by c, first 2^{k-1} leaves are labeled by a, and remaining 2^{k-1} leaves are labeled by b.

In this example we are going to derive the following types:

$$\begin{aligned} \hat{\sigma}_\mathsf{a} &= (\{(0,\mathsf{a})\}, \mathbf{0}, o)\,, & \hat{\rho}_1^\Delta &= (\emptyset, \{\!|0,0,1,1|\!\}, o)\,, \\ \hat{\sigma}_\mathsf{b} &= (\{(0,\mathsf{b})\}, \mathbf{0}, o)\,, & \hat{\tau}_\mathsf{f} &= (\emptyset, \{\!|1,1|\!\}, \{\!|\hat{\sigma}_\mathsf{a}, \hat{\sigma}_\mathsf{m}|\!\} \to \{\!|\hat{\sigma}_\mathsf{b}, \hat{\sigma}_\mathsf{m}|\!\} \to o)\,, \\ \hat{\sigma}_\mathsf{m} &= (\emptyset, \{\!|0|\!\}, o)\,, & \hat{\sigma}_R^\Delta &= (\emptyset, \{\!|0,0|\!\}, \{\!|\hat{\tau}_\mathsf{f}|\!\} \to o)\,. \end{aligned}$$

Denote subterms of $\Lambda(\mathcal{G}_4)$ as follows:

$$S = \mathsf{f}\,(\mathsf{a}\langle\rangle)\,(\mathsf{b}\langle\rangle)\,, \qquad T = \lambda \mathsf{x}.\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{c}\langle \mathsf{x}, \mathsf{x}\rangle)\,(\mathsf{c}\langle \mathsf{y}, \mathsf{y}\rangle)\,, \qquad R_4 = \lambda \mathsf{f}.\mathsf{nd}\langle S, R_4\,T\rangle\,.$$

We first derive the type triple $\hat{\sigma}_R^\Delta$ for R_4 with empty flag counter:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{f}|\!\}] \vdash_1^\Delta \mathsf{f} : \hat{\tau}_\mathsf{f} \triangleright \mathbf{0} \\ \varepsilon \vdash_1^\Delta \mathsf{a}\langle\rangle : \hat{\sigma}_\mathsf{a} \triangleright \mathbf{0} \qquad \varepsilon \vdash_1^\Delta \mathsf{a}\langle\rangle : \hat{\sigma}_\mathsf{m} \triangleright \mathbf{0} \qquad \varepsilon \vdash_1^\Delta \mathsf{b}\langle\rangle : \hat{\sigma}_\mathsf{b} \triangleright \mathbf{0} \qquad \varepsilon \vdash_1^\Delta \mathsf{b}\langle\rangle : \hat{\sigma}_\mathsf{m} \triangleright \mathbf{0}\end{array}}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{f}|\!\}] \vdash_1^\Delta S : \hat{\rho}_1^\Delta \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!|\hat{\tau}_\mathsf{f}|\!\}] \vdash_1^\Delta \mathsf{nd}\langle S, R_4\,T\rangle : \hat{\rho}_1^\Delta \triangleright \mathbf{0}}\,(\textsc{Nd})}{\varepsilon \vdash_1^\Delta R_4 : \hat{\sigma}_R^\Delta \triangleright \mathbf{0}}\,(\lambda)$$

Here the first premiss of the (@) rule can be derived by the (Var) rule, and the other premisses by the (Con0) rule. Notice that while deriving $\varepsilon \vdash_1^\Delta \mathsf{a}\langle\rangle : \hat{\sigma}_\mathsf{m} \triangleright \mathbf{0}$ a $(1,\mathsf{a})$-flag is created by the *Comp* predicate; we are, however, not obliged to store the information about it in the flag set, and thus we can derive the type triple $\hat{\sigma}_\mathsf{m}$ instead of $(\{(1,\mathsf{a})\}, \{\!|0|\!\}, o)$. Notice also that the (@) rule have premisses with $(0,\mathsf{a})$ in the flag set, and with 0 and 1 in the marker multiset. Passing them all to the *Comp* predicate would result in creating a $(2,\mathsf{a})$-flag and increasing the flag counter. This is not the case, because the (@) rule does not pass $(0,\mathsf{a})$ to *Comp*, because 0 is not greater than $\mathit{ord}(\mathsf{a}\langle\rangle)=0$.

Next, we would like to increase the flag counter, by using the second child of the $\mathsf{nd}\langle\cdot,\cdot\rangle$ node constructor. We start as follows:

$$\dfrac{\dfrac{}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}}|\!\}]\vdash^{\triangle}_{1}\mathsf{x}:\hat\sigma_{\mathsf{a}}\triangleright\mathbf{0}}\,(\textsc{Var})\qquad\dfrac{}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}}|\!\}]\vdash^{\triangle}_{1}\mathsf{x}:\hat\sigma_{\mathsf{a}}\triangleright\mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}}|\!\}]\vdash^{\triangle}_{1}\mathsf{c}\langle\mathsf{x},\mathsf{x}\rangle:\hat\sigma_{\mathsf{a}}\triangleright\mathbf{0}}\,(\textsc{Con}{\geq}1)$$

$$\dfrac{\dfrac{}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}}|\!\}]\vdash^{\triangle}_{1}\mathsf{x}:\hat\sigma_{\mathsf{a}}\triangleright\mathbf{0}}\,(\textsc{Var})\qquad\dfrac{}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{x}:\hat\sigma_{\mathsf{m}}\triangleright\mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}},\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{c}\langle\mathsf{x},\mathsf{x}\rangle:(\{(1,\mathsf{a})\},\{\!|0|\!\},o)\triangleright\mathbf{0}}\,(\textsc{Con}{\geq}1)$$

In the latter derivation, the $(\textsc{Con}{\geq}1)$ rule created a $(1,\mathsf{a})$-flag because the information about a $(0,\mathsf{a})$-flag from the first premiss has met the information about an order-0 marker from the second premiss. Let us denote the conclusions of the above derivations by $J_{\mathsf{x},\mathsf{a}}$ and $J_{\mathsf{x},\mathsf{m}}$, respectively. Similarly we can derive $\varepsilon[\mathsf{y}\mapsto\{\!|\hat\sigma_{\mathsf{b}}|\!\}]\vdash^{\triangle}_{1}\mathsf{c}\langle\mathsf{y},\mathsf{y}\rangle:\hat\sigma_{\mathsf{b}}\triangleright\mathbf{0}$, denoted $J_{\mathsf{y},\mathsf{b}}$, and $\varepsilon[\mathsf{y}\mapsto\{\!|\hat\sigma_{\mathsf{b}},\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{c}\langle\mathsf{y},\mathsf{y}\rangle:(\{(1,\mathsf{b})\},\{\!|0|\!\},o)\triangleright\mathbf{0}$, denoted $J_{\mathsf{y},\mathsf{m}}$. Denote also $c_1=\{(\mathsf{a},1),(\mathsf{b},1)\}$. We continue as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\}]\vdash^{\triangle}_{1}\mathsf{f}:\hat\tau_{\mathsf{f}}\triangleright\mathbf{0}}\,(\textsc{Var})\qquad J_{\mathsf{x},\mathsf{a}}\qquad J_{\mathsf{x},\mathsf{m}}}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\},\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}},\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{f}\,(\mathsf{c}\langle\mathsf{x},\mathsf{x}\rangle):(\emptyset,\{\!|0,1,1|\!\},\{\!|\hat\sigma_{\mathsf{b}},\hat\sigma_{\mathsf{m}}|\!\}{\to}o)\triangleright c_{\mathsf{a}}}\,(@)\qquad J_{\mathsf{y},\mathsf{b}}\qquad J_{\mathsf{y},\mathsf{m}}}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\},\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}},\hat\sigma_{\mathsf{m}}|\!\},\mathsf{y}\mapsto\{\!|\hat\sigma_{\mathsf{b}},\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{f}\,(\mathsf{c}\langle\mathsf{x},\mathsf{x}\rangle)\,(\mathsf{c}\langle\mathsf{y},\mathsf{y}\rangle):(\emptyset,\{\!|0,0,1,1|\!\},o)\triangleright c_1}\,(@)}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\},\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{a}},\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{c}\langle\mathsf{x},\mathsf{x}\rangle)\,(\mathsf{c}\langle\mathsf{y},\mathsf{y}\rangle):(\emptyset,\{\!|0,1,1|\!\},\{\hat\sigma_{\mathsf{b}},\hat\sigma_{\mathsf{m}}\}{\to}o)\triangleright c_1}\,(@)}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\}]\vdash^{\triangle}_{1}T:\hat\tau_{\mathsf{f}}\triangleright c_1}\,(\lambda)$$

In the (@) rules, the information about order-1 flags from $J_{\mathsf{x},\mathsf{a}}$ and $J_{\mathsf{y},\mathsf{b}}$ meets the information about order-1 markers, and thus order-2 flags are created, which results in increasing the flag counter. Finally, we can derive:

$$\dfrac{\dfrac{\dfrac{\varepsilon\vdash^{\triangle}_{1}R_4:\hat\sigma^{\triangle}_{R}\triangleright c\qquad\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\}]\vdash^{\triangle}_{1}T:\hat\tau_{\mathsf{f}}\triangleright c_1}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\}]\vdash^{\triangle}_{1}R_4\,T:\hat\rho^{\triangle}_{1}\triangleright c+c_1}\,(@)}{\varepsilon[\mathsf{f}\mapsto\{\!|\hat\tau_{\mathsf{f}}|\!\}]\vdash^{\triangle}_{1}\mathsf{nd}\langle S,R_4\,T\rangle:\hat\rho^{\triangle}_{1}\triangleright c+c_1}\,(\textsc{Nd})}{\varepsilon\vdash^{\triangle}_{1}R_4:\hat\sigma^{\triangle}_{R}\triangleright c+c_1}\,(\lambda)$$

By repeating this derivation fragment some number of times, we can derive $\hat\sigma^{\triangle}_{R}$ for R_4 with arbitrarily large flag counter c (satisfying $c(\mathsf{a})=c(\mathsf{b})$).

We also need to derive the type triple $\hat\tau_{\mathsf{f}}$ for the subterm $\lambda\mathsf{x}.\lambda\mathsf{y}.\mathsf{c}\langle\mathsf{x},\mathsf{y}\rangle$. This is done as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{x}:(\emptyset,\{\!|0,1,1|\!\},o)\triangleright\mathbf{0}}\,(\textsc{Var})\qquad\dfrac{}{\varepsilon[\mathsf{y}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{y}:\hat\sigma_{\mathsf{m}}\triangleright\mathbf{0}}\,(\textsc{Var})}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\},\mathsf{y}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\mathsf{c}\langle\mathsf{x},\mathsf{y}\rangle:\hat\rho^{\triangle}_{1}\triangleright\mathbf{0}}\,(\textsc{Con}{\geq}1)}{\varepsilon[\mathsf{x}\mapsto\{\!|\hat\sigma_{\mathsf{m}}|\!\}]\vdash^{\triangle}_{1}\lambda\mathsf{y}.\mathsf{c}\langle\mathsf{x},\mathsf{y}\rangle:(\emptyset,\{\!|0,1,1|\!\},\{\!|\hat\sigma_{\mathsf{b}},\hat\sigma_{\mathsf{m}}|\!\}{\to}o)\triangleright\mathbf{0}}\,(\lambda)}{\varepsilon\vdash^{\triangle}_{1}\lambda\mathsf{x}.\lambda\mathsf{y}.\mathsf{c}\langle\mathsf{x},\mathsf{y}\rangle:\hat\tau_{\mathsf{f}}\triangleright\mathbf{0}}\,(\lambda)$$

Above, we have put two order-1 markers in the leaf describing the subterm x. We could equally well put them in the leaf corresponding to y, or one marker here and one there.

This allows us to finish by applying the (@) rule:

$$\dfrac{\varepsilon\vdash^{\triangle}_{1}R_4:\hat\sigma^{\triangle}_{R}\triangleright c\qquad\varepsilon\vdash^{\triangle}_{1}\lambda\mathsf{x}.\lambda\mathsf{y}.\mathsf{c}\langle\mathsf{x},\mathsf{y}\rangle:\hat\tau_{\mathsf{f}}\triangleright\mathbf{0}}{\varepsilon\vdash^{\triangle}_{1}\Lambda(\mathcal{G}_4):\hat\rho^{\triangle}_{1}\triangleright c}\,(@)$$

F Finite Prefixes of Infinite λ-terms

Theorems 3 and D.1 talk about infinite λ-terms, but the properties described by these theorems concern actually only finite prefixes of these λ-terms. Moreover, while proving these two theorems it is easier to concentrate on finite λ-terms. For this reason we now formalize the concept of taking a finite prefix of a λ-term.

We first say what does it mean that one λ-term is a prefix of another λ-term. This is described by the relation $\preccurlyeq$ defined as the smallest reflexive relation such that:

- $\lambda x_1^{\alpha_1}.\cdots.\lambda x_s^{\alpha_s}.\mathsf{nd}\langle\rangle \preccurlyeq Q$ 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 λ-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 λ-term remains unchanged).

The fact that in Theorems 3 and D.1 it is enough to consider finite prefixes of λ-terms is given by the following two lemmata.

▶ **Lemma F.1.** *We can derive a type judgment $\Gamma\vdash_m^\kappa P:\hat\tau\triangleright c$ if and only if for some finite λ-term P' such that $P'\preccurlyeq P$ we can derive $\Gamma\vdash_m^\kappa P':\hat\tau\triangleright c$.*

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 $\Gamma\vdash_m^\kappa P:\hat\tau\triangleright c$. For the opposite implication we observe that it is impossible to derive any type judgment for a λ-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 these form by the actual subterms of P, without altering the type derivation. Details, being easy, are left to the reader. ◀

▶ **Lemma F.2.** *Let P be a closed λ-term of sort o. For every tree T it holds that $T\in\mathcal{L}(BT(P))$ if and only if there exists a finite λ-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 the above lemma. The first three lemmata are useful while showing its right-to-left implication.

▶ **Lemma F.3.** *Suppose that $R'\preccurlyeq R$ and $S'\preccurlyeq S$, where R' is finite.*[4] *Then $R'[S'/x]\preccurlyeq R[S/x]$.*

Proof. A trivial induction on the size of R'. ◀

▶ **Lemma F.4.** *Suppose that $P'\preccurlyeq P$ and $P'\to_\beta^* Q'$, where P' is finite. Then there exists a λ-term Q such that $Q'\preccurlyeq Q$ and $P\to_\beta^* Q$.*

Proof. We proceed by induction on the length of the shortest reduction sequence witnessing $P'\to_\beta^* Q'$; only the base case of a single β-reduction is interesting, thus assume that $P'\to_\beta Q'$. Internally, we proceed by induction on the depth of the β-reduction $P'\to_\beta Q'$. Again, only the base case is interesting, thus assume that $P'=(\lambda x.R')\,S'$ and $Q'=R'[S'/x]$.

We have two cases. One possibility is that $P=(\lambda x.R)\,S$, where $R'\preccurlyeq R$ and $S'\preccurlyeq S$. In this case, taking $Q=R[S/x]$ we have that $Q'\preccurlyeq Q$, and, by Lemma F.3, $Q'\preccurlyeq Q$.

[4] It is convenient to proceed by induction on the size of R', and thus we assume that it is finite, but equally well the lemma could be shown without this assumption.

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 $Q' = R' \preccurlyeq Q$.[5] ◀

▶ **Lemma F.5.** *Let Q' and Q be closed λ-terms of sort o such that $Q' \preccurlyeq Q$, and let T be a finite Σ-labeled tree such that $Q' \rightarrow^n_{\mathsf{nd}} T$. Then $BT(Q) \rightarrow^*_{\mathsf{nd}} T$.*

Proof. The proof is by induction on $|T|+n$. Since $Q' \rightarrow^n_{\mathsf{nd}} T$, necessarily $Q' = a\langle Q'_1,\dots,Q'_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$. Since $Q' \preccurlyeq Q$, we also have that $Q = a\langle Q_1,\dots,Q_r\rangle$, and thus $BT(Q) = a\langle BT(Q_1),\dots,BT(Q_r)\rangle$, where $Q'_i \preccurlyeq Q_i$ for all $i \in \{1,\dots,r\}$. We have two cases.

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

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

The first step needed while proving the left-to-right implication of Lemma F.2 is to show that every tree from $\mathcal{L}(BT(P))$ can be seen already after performing finitely many β-reductions from P.

▶ **Lemma F.6.** *Let P be a closed λ-term of sort o, and let T be a finite Σ-labeled tree such that $BT(P) \rightarrow^n_{\mathsf{nd}} T$. Then there exists a λ-term Q such that $P \rightarrow^*_\beta Q \rightarrow^*_{\mathsf{nd}} T$.*

Proof. The proof is by induction on $|T|+n$. Since $BT(P) \rightarrow^n_{\mathsf{nd}} T$, necessarily $BT(P) \neq \mathsf{nd}\langle\rangle$, and thus $P \rightarrow^*_\beta a\langle P_1,\dots,P_r\rangle$ for some $a \in \Sigma^{\mathsf{nd}}$ and some λ-terms $P_1,\dots,P_r$ such 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 $T = a\langle T_1,\dots,T_r\rangle$, and for all $i \in \{1,\dots,r\}$ it holds that $|T_i| < |T|$ and $BT(P_i) \rightarrow^{n_i}_{\mathsf{nd}} T_i$ for some $n_i \le n$. For every $i \in \{1,\dots,r\}$ the induction assumption gives us a λ-term Q_i such that $P_i \rightarrow^*_\beta Q_i \rightarrow^*_{\mathsf{nd}} T_i$. Taking $Q = a\langle Q_1,\dots,Q_r\rangle$ we obtain $P \rightarrow^*_\beta Q \rightarrow^*_{\mathsf{nd}} T$, as required.

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

It is convenient to introduce one more relation: we write $P \approx_l P'$ if the λ-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 λ-terms P, Q of the same sort,
- $a\langle P_1,\dots,P_r\rangle \approx_l a\langle P'_1,\dots,P'_r\rangle$ if $l > 0$ and $P_i \approx_{l-1} P'_i$ for all $i \in \{1,\dots,r\}$,
- $P\,Q \approx_l P'\,Q'$ 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 λ-term Q obtained in Lemma F.6 is important.

▶ **Lemma F.7.** *Let Q be a closed λ-term of sort o, and let T be a finite Σ-labeled tree such that $Q \rightarrow^n_{\mathsf{nd}} T$. Then $Q' \rightarrow^*_{\mathsf{nd}} T$ for all λ-terms Q' such that $Q \approx_{|T|+n} Q'$.*

[5] In the latter case we only know that R is of the form $T\,S$, but not necessarily $(\lambda x.R)\,S$, so we cannot proceed as in the former case.

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 $a\langle Q_1,\dots,Q_r\rangle$ and $a\langle Q_1',\dots,Q_r'\rangle$, respectively, where $Q_i \approx_{|T|+n-1} Q_i'$ for all $i \in \{1,\dots,r\}$. We have two cases.

Suppose first that $a \neq \mathsf{nd}$. Then $T = a\langle T_1,\dots,T_r\rangle$, and for all $i \in \{1,\dots,r\}$ it holds 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 $Q_i' \to_{\mathsf{nd}}^{*} T_i$ by the induction assumption (for all $i \in \{1,\dots,r\}$). In consequence $Q' \to_{\mathsf{nd}}^{*} T$.

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

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

▶ **Lemma F.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 $P = a\langle P_1,\dots,P_r\rangle$ or $P = \lambda y.Q$ are similar. Finally, when $P = P'$ is a variable, the thesis follows immediately from $Q \approx_l Q'$. ◀

▶ **Lemma F.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 F.8, $Q \approx_l Q'$. The remaining case is that $l > 0$ and the redex involved in the β-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 F.2.

Proof of Lemma F.2. Let us first establish the right-to-left implication. We assume here that $P' \preccurlyeq P$ and $T \in \mathcal{L}(BT(P'))$ for a finite λ-term P', and we need to prove that $T \in \mathcal{L}(BT(P))$. Denote $Q' = BT(P')$. Since P' is finite, we have $P' \to_\beta^* Q'$ (the Böhm tree of a finite λ-term is just its β-normal form). Lemma F.4 gives us a λ-term Q such that $Q' \preccurlyeq Q$ and $P \to_\beta^* Q$. Since $T \in \mathcal{L}(BT(P'))$, by definition of $\mathcal{L}(\cdot)$, we have that $Q' \to_{\mathsf{nd}}^{n} T$ for some $n \in \mathbb{N}$, and that T is a finite Σ-labeled tree. In such a situation Lemma F.5 implies $BT(Q) \to_{\mathsf{nd}}^{*} T$. Since $BT(P) = BT(Q)$, we obtain $T \in \mathcal{L}(BT(P))$, as required.

Let us now prove the opposite implication. We know that $T \in \mathcal{L}(BT(P))$, i.e., 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 F.6, there exists a λ-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 λ-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 F.9 consecutively to every β-reduction in the reduction sequence witnessing $P \to_\beta^k Q$ we obtain a λ-term Q' such that $P' \to_\beta^* Q'$ and $Q \approx_{m+|T|} Q'$. Next, Lemma F.7 implies that $Q' \to_{\mathsf{nd}}^{*} T$. Finally, we use Lemma F.5, where we set both Q and Q' to Q'; it gives us that $BT(Q') \to_{\mathsf{nd}}^{*} T$. Since $BT(P') = BT(Q')$, we obtain $T \in \mathcal{L}(BT(P'))$, as required. ◀

G Properties of Type Judgments

Before actually proving Theorems 3 and D.1 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 observation, that follows directly from rules of the type system. This observation will be used implicitly later.

▶ **Observation.** *If we can derive $\Gamma \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright c$, and x is not free in R, then $\Gamma(x) = \mathbf{0}$.*

Next, in Lemma G.1, 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 G.1.** *Suppose that we can derive $\Gamma \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright c$. Then $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$.*

Proof. Fix some derivation of $\Gamma \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright c$; 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 $\mathsf{Mk}(\hat{\tau}') \leq \mathsf{Mk}(\hat{\tau})$, which implies that $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$.

In the case when $R = a\langle\rangle$ for $a \neq \mathsf{nd}$, the (Con0) rule implies that $\Gamma = \varepsilon$, hence $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$.

Next, suppose that $R = \lambda x.P$. Let $\Gamma[x \mapsto C'] \vdash^{\kappa}_{m} P : \hat{\tau}' \triangleright c$ 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 $\mathsf{Mk}(\Gamma[x \mapsto C']) \leq \mathsf{Mk}(\hat{\tau}')$. Putting this together we obtain $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$.

Finally, suppose that $R = a\langle P_1, \dots, P_r\rangle$ where $r \geq 1$ or $a = \mathsf{nd}$, 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 (Con≥1), or (Nd), or (@)), and let $\Gamma_1, \dots, \Gamma_s$ be the type environments used there. Each of the four possible rules ensures that $\mathsf{Mk}(\hat{\tau}) = \mathsf{Mk}(\hat{\tau}_1) + \dots + \mathsf{Mk}(\hat{\tau}_s)$ and $\Gamma = \Gamma_1 \sqcup \dots \sqcup \Gamma_s$. The induction assumption gives us inequalities $\mathsf{Mk}(\Gamma_i) \leq \mathsf{Mk}(\hat{\tau}_i)$ for all $i \in \{1, \dots, s\}$. It follows that $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$. ◀

The next important property of our type system is given in Lemma G.2.

▶ **Lemma G.2.** *If a type judgment $\Delta \vdash^{\kappa}_{m} R : \hat{\sigma} \triangleright d$ is used in a derivation of $\Gamma \vdash^{\kappa}_{m} S : \hat{\tau} \triangleright c$, 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 $\hat{\sigma} = (F, M, C_1{\to}\dots{\to}C_s{\to}o)$ is *m-clear* if it holds that $(M + \sum_{i=1}^{s} \mathsf{Mk}(C_i))(m) = 0$. 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 λ-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 $\hat{\sigma} = (F, M, C_1{\to}\dots{\to}C_s{\to}o)$. For $i \in \{1, \dots, s\}$ by definition we have $C_i \subseteq \mathcal{TT}^{\alpha_i}_{ord(\alpha_i)}$, where $\alpha = \alpha_1{\to}\dots{\to}\alpha_s{\to}o$; thus $\mathsf{Mk}(C_i)$ 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 $\Delta \vdash^{\kappa}_{m} R : (F, M, \sigma) \triangleright d$ be the considered conclusion, where $\sigma = C_1{\to}\dots{\to}C_s{\to}o$. 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 $R = \mathsf{nd}\langle P_1, \dots, P_r\rangle$, then the (ND) rule is used, so the type triple derived in the premiss is the same as in the conclusion.

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

Next, suppose that $R = a\langle P_1, \dots, P_r\rangle$, where $a \neq \mathsf{nd}$ and $r \geq 1$. Then the premisses are $\Delta_i \vdash^{\kappa}_{m} P_i : (F_i, M_i, o) \triangleright d_i$ for $i \in \{1, \dots, r\}$, where the rule, being either (CON1) or (CON$\geq$1), ensures that $M = M_1 + \dots + M_r$. We thus immediately have $M_i(m) \leq M(m) = 0$ for all $i \in \{1, \dots, r\}$.

Finally, suppose that $R = P\,Q$. Let $\Delta' \vdash^{\kappa}_{m} P : (F', M', C_0 \to \sigma) \triangleright d'$ and $\Delta_i \vdash^{\kappa}_{m} Q : (F_i, M_i, \sigma_i) \triangleright d_i$ 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 G.2 we easily deduce the following lemma.

▶ **Lemma G.3.** *Suppose that we can derive $\Gamma \vdash^{\kappa}_{m} S : \hat{\tau} \triangleright c$, 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 $\Delta \vdash^{\kappa}_{m} R : (F, M, \sigma) \triangleright d$ 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), (CON$\geq$1), or (@). In these rules we have $(F, d) \in \mathit{Comp}_m(M; \cdots)$, where by Lemma G.2 we have $M(m) = 0$. Moreover, in all pairs (F_i, c_i) passed to this Comp_m predicate we have that $c_i = \mathbf{0}$ and that F_i is m-bounded (this is also the case for $F_i = \{(0, a)\}$ since $m \geq ord(S) \geq 0$). We see 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 G.3 can be generalized to lower orders, i.e., that whenever we can derive $\Gamma \vdash^{\kappa}_{m} S : (F, M, \tau) \triangleright c$ 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 G.3: 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 λ-term. Live 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 G.1.** In this example we suppose that $A = \{\mathsf{a}\}$; then $\hat{\rho}_1 = (\emptyset, \{\!|0, 1|\!\}, o)$. 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 $(o \to o) \to o \to o$.

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

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 λ-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 G.3 in a more subtle way, having in mind the above issues. To this end, we proceed in a minimalistic way: in Lemma G.4 we prove what will be 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 $\Gamma \vdash^{\kappa}_{m} P\,Q : \hat{\tau} \triangleright c$. We say that this use of the (@) rule is *wild* if it has a premiss $\Gamma' \vdash^{\kappa}_{m} Q : (F, \mathbf{0}, \sigma) \triangleright c'$ such that for some $(k, a) \in F{\restriction}_{>ord(Q)}$ it holds that $\mathsf{Mk}(\hat{\tau})(l) > 0$ for all $l \in \{k, k+1, \dots, m\}$. A type derivation is *wild* if, at some moment, it uses the (@) rule in a wild way.

▶ **Lemma G.4.** *There is no wild derivation of $\Gamma \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$, where P is a closed λ-term of sort o and complexity at most $m+1$.*

This lemma will be proven in Appendix I. We remark that we do not use this lemma in Appendices H and I, only in Appendix J. Right now we only prove the following auxiliary lemma.

▶ **Lemma G.5.** *There is no wild derivation of $\Gamma \vdash^{\kappa}_{m} S : \hat{\tau} \triangleright c$ 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 $\Delta \vdash^{\kappa}_{m} P\,Q : \hat{\sigma} \triangleright d$ be the conclusion of this rule. The wildness condition requires in particular that $\mathsf{Mk}(\hat{\sigma})(m) > 0$, but by Lemma G.2 we have $\mathsf{Mk}(\hat{\sigma})(m) = 0$, so all this could not happen. ◀

H Completeness

In this section we prove the left-to-right implication of Theorems 3 and D.1. We divide the proof into the following four lemmata. Recall that $P \to_{\beta(k)} Q$ denotes a β-reduction of order k, i.e., concerning a redex $(\lambda x.R)\,S$ with $ord(x) = k$.

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

▶ **Lemma H.2.** *Suppose that $T \in \mathcal{L}(P)$, and that $c\colon A \to \mathbb{N}$ is such that for every $a \in A$, $c(a)$ is the number of appearances of a in T. Then we can derive $\varepsilon \vdash^{\Delta}_{-1} P : \hat{\rho}^{\Delta}_{-1} \triangleright c$. Moreover, if T is a word, then we can derive $\varepsilon \vdash_{0} P : \hat{\rho}_{0} \triangleright c$.*

▶ **Lemma H.3.** *Suppose that $P \to_{\beta(m)} Q$, where $m \geq 0$. If we can derive $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright c$, then we can also derive $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$.*

▶ **Lemma H.4.** *If we can derive $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$ (where $m \geq 0$ if $\kappa = \varepsilon$, and $m \geq -1$ if $\kappa = \Delta$), then we can also derive $\varepsilon \vdash^{\kappa}_{m+1} P : \hat{\rho}^{\kappa}_{m+1} \triangleright c'$ for some c' such that $c'(a) \geq \left\lfloor \frac{1}{|A|} \log_2 c(a) \right\rfloor$ for all $a \in A$.*

Now the left-to-right implication of Theorem 3 easily follows. Indeed, take a closed word-recognizing λ-term P of sort o and complexity at most $m+1$ such that $Diag_A(\mathcal{L}(BT(P)))$ holds, and take any $n \in \mathbb{N}$. Let us denote $f_0(l) = l$ and $f_{k+1}(l) = \left\lfloor \frac{1}{|A|} \log_2 f_k(l) \right\rfloor$. We can find a number n' such that $f_m(n') \geq n$, as well as a tree $T \in \mathcal{L}(BT(P))$ such that every symbol from A appears in T at least n' times. We first apply Lemma F.2, obtaining a finite λ-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 H.1 to P', obtaining λ-terms $Q_{m+1}, Q_m, \dots, Q_0$ with $T \in \mathcal{L}(Q_0) = \mathcal{L}(BT(P'))$. Since T is actually a word, by Lemma H.2 we can derive $\varepsilon \vdash_0 Q_0 : \hat{\rho}_0 \triangleright c_0$ with $c_0(a) \geq n' = f_0(n')$ for all $a \in A$. Then for every $k \in \{0, \dots, m\}$ we perform two steps. First, we repeatedly apply Lemma H.3 to every β-reduction (of order k) between Q_{k+1} and Q_k, obtaining a derivation of $\varepsilon \vdash_k Q_{k+1} : \hat{\rho}_k \triangleright c_k$. Then, if $k < m$, we apply Lemma H.4, obtaining a derivation of $\varepsilon \vdash_{k+1} Q_{k+1} : \hat{\rho}_{k+1} \triangleright c_{k+1}$ for some c_{k+1} such that $c_{k+1}(a) \geq f_1(f_k(n')) = f_{k+1}(n')$ for all $a \in A$. We end up with a derivation of $\varepsilon \vdash_m P' : \hat{\rho}_m \triangleright c_m$, where $c_m(a) \geq f_m(n') \geq n$ for all $a \in A$. Using Lemma F.1 we can convert it into a derivation of $\varepsilon \vdash_m P : \hat{\rho}_m \triangleright c_m$, as needed.

Similarly we prove the left-to-right implication of Theorem D.1, where the λ-term P need not to be word-recognizing. We start as previously, but this time we take n' such that $f_{m+1}(n') \geq n$. The proofs start to diverge when we use Lemma H.2: this time T is not necessarily a tree, thus we obtain a derivation of $\varepsilon \vdash^{\Delta}_{-1} Q_0 : \hat{\rho}^{\Delta}_{-1} \triangleright c_{-1}$ with $c_{-1}(a) \geq n'$ for all $a \in A$. We then apply Lemma H.4, obtaining a derivation of $\varepsilon \vdash^{\Delta}_0 Q_0 : \hat{\rho}^{\Delta}_0 \triangleright c_0$ for some c_0 such that $c_0(a) \geq f_1(n')$ for all $a \in A$. We continue as for words, alternatingly applying Lemmata H.3 and H.4, so that we end up with a derivation of $\varepsilon \vdash^{\Delta}_m P : \hat{\rho}^{\Delta}_m \triangleright c_m$, where $c_m(a) \geq f_m(f_1(n')) \geq n$ for all $a \in A$, as needed.

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

H.1 Proof of Lemma H.1

Lemma H.1 is a consequence of the following lemma.

▶ **Lemma H.5.** *Let P be a finite closed λ-term of sort o and complexity $n > 0$. Then $P \to_{\beta(n-1)} Q$ for some λ-term Q.*

Proof. Let R_0 be a minimal subterm of P that is of order smaller than n, but has complexity n (i.e., has subterms of order n). This minimality implies that some "direct subterm" of R_0 has order n. This is possible only when $R_0 = R_1\, S_1$ is an application, where $ord(R_1) = n$. Because of the order, R_1 cannot be a node constructor. Moreover, R_1 cannot be a variable as well: since P is closed, this variable has to be bound by some lambda in P; but then the subterm starting by this lambda would be of order at least $n+1$ contradicting the fact that P has complexity n. Thus R_1 is a λ-binder or an application. If $R_1 = R_2\, S_2$ is an application, then $ord(R_2) = n$, and the same argument shows that R_2 is a λ-binder or an application. Continuing this way, we obtain that $R_0 = (\lambda x.R')\, S_s \dots S_1$ for some $s \geq 1$. Then, by homogeneity of the sort of $\lambda x.R'$, we have that $ord(x) = n-1$ (the first argument cannot be of smaller order than any of the next arguments). Thus the β-reduction that replaces in P the subterm $(\lambda x.R')\, S_s$ by $R'[S_s/x]$ is of order $n-1$. ◀

Proof of Lemma H.1. Recall that we are given a closed λ-term P of sort o and complexity at most n. We take $Q_n = P$. Then, for every $k = n, n-1, \dots, 1$, we start performing β-reductions of order $k-1$ from Q_k, and as Q_{k-1} we take the obtained λ-term from which no more β-reductions of order $k-1$ can be performed. Since Q_k is finite, the process necessarily stops. The complexity of a λ-term does not grow during β-reductions, thus by Lemma H.5, the complexity of Q_{k-1} is at most $k-1$. In particular the complexity of Q_0 is 0, and thus $Q_0 = BT(P)$. ◀

▶ Remark. We notice that Lemma H.1 would be false if we have allowed λ-terms involving non-homogeneous sorts. For example, in a λ-term of the form $(\lambda x.\lambda y.P)\,Q\,R$ with $ord(x) = 0$ and $ord(y) = 1$ we have to perform a β-reduction of order 0 concerning x before a β-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 H.1 would work also for λ-terms involving non-homogeneous sorts if we have defined the order of a β-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 a contextual information, not determined by x itself. For this reason we prefer to restrict ourselves to homogeneous sorts.

H.2 Proof of Lemma H.2

Basically, we just apply the rules of our type system, namely the rules (Con0), (Con1) (or (Con≥1) in the tree case), and (Nd), in every nd-labeled node choosing the subtree in which T continues; then the flag counter will compute exactly the number of appearances of every symbol from A 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 $n \in \mathbb{N}$). Take $\kappa = \triangle$ and $m = -1$ while proving the first statement, and $\kappa = \varepsilon$, $m = 0$ in the case of T being a word. In both cases, we intend to derive $\varepsilon \vdash_m^\kappa P : \hat\rho_m^\kappa \triangleright c$.

We have two 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 $\varepsilon \vdash_m^\kappa P_i : \hat\rho_m^\kappa \triangleright c$, where, for all $a \in A$, $c(a)$ is the number of appearances of a in T. To this type judgment we apply the (Nd) rule, deriving $\varepsilon \vdash_m^\kappa P : \hat\rho_m^\kappa \triangleright c$, as needed.

Because $P \to_{\mathsf{nd}}^* T$, and T is a tree, the only remaining case is that $P = b\langle P_1, \dots, P_r\rangle$ for some $b \in \Sigma$. Then necessarily $T = b\langle T_1, \dots, T_r\rangle$, and out of the reduction sequence witnessing $P \to_{\mathsf{nd}}^n T$ we can extract a reduction sequences witnessing $P_i \to_{\mathsf{nd}}^{n_i} T_i$ for every $i \in \{1, \dots, r\}$, where $n_i \le n$. For $i \in \{1, \dots, r\}$, by the induction assumption we know that we can derive $\varepsilon \vdash_m^\kappa P_i : \hat\rho_m^\kappa \triangleright c_i$, where $c_i \colon A \to \mathbb{N}$ is such that, for all $a \in A$, $c_i(a)$ is the number of appearances of a in T_i. Obviously, $c(a) = c_1(a) + \dots + c_r(a)$ for $a \in A \setminus \{b\}$, and $c(a) = 1 + c_1(a) + \dots + c_r(a)$ if $a = b \in A$. Directly from the definition of the *Comp* predicate it follows that

$$\begin{aligned}
&(\emptyset, c) \in Comp_{-1}(\mathbf{0}; (\{(0,b)\}, \mathbf{0}), (\emptyset, c_1), \dots, (\emptyset, c_r))\,, && \text{and}\\
&(\emptyset, c) \in Comp_0(\{\!\{0\}\!\}; (\{(0,b)\}, \mathbf{0}), (\emptyset, c_1), \dots, (\emptyset, c_r))\,, && \text{hence more generally}\\
&(\emptyset, c) \in Comp_m(\mathsf{Mk}(\hat\rho_m^\kappa); (\{(0,b)\}, \mathbf{0}), (\emptyset, c_1), \dots, (\emptyset, c_r))\,.
\end{aligned}$$

We recall here that $\mathsf{Mk}(\hat\rho_{-1}^\triangle) = \mathbf{0}$ and $\mathsf{Mk}(\hat\rho_0) = \{\!\{0\}\!\}$; in both cases the flag set in $\hat\rho_m^\kappa$ is $\emptyset$.

If $r = 0$, we simply apply the (Con0) rule. If $r = 1$ and $\kappa = \varepsilon$, we apply the (Con1) rule. Recall that for $\kappa = \varepsilon$ we assume that T is a tree, which means that $r \le 1$. Thus the remaining case is that $r \ge 1$ and $\kappa = \triangle$. In this case we apply the (Con≥1) rule, where we need to notice that $\mathsf{Mk}(\rho^{\triangle}_{-1}) + \dots + \mathsf{Mk}(\rho^{\triangle}_{-1}) = \mathbf{0} = \mathsf{Mk}(\rho^{\triangle}_{-1})$. In all cases, the derived type judgment is $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$, as needed.

H.3 Proof of Lemma H.3

Before going into details, let us sketch the proof. Knowing that we can derive $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright c$, we want to derive the same for a given λ-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 $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright c$ we identify the set I of places (nodes) where we derive a type for S substituted for x. For $i \in I$, let $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ be the type judgment in i. We change the nodes in I into leaves, where we instead derive $\varepsilon[x \mapsto \{\!|\hat{\sigma}_i|\!\}] \vdash^{\kappa}_{m} x : \hat{\sigma}_i \triangleright \mathbf{0}$ (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 λ-terms, and decreasing flag counters. In this way we obtain derivations of $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and a derivation of $\Upsilon^x \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$, where[6] $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$, 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 $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$ (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 also need to know that $\{\!|\hat{\sigma}_i \mid i \in I|\!\}$ is indeed a triple container, i.e., that every unbalanced type triple appears as $\hat{\sigma}_i$ for at most $|A|$ indices $i \in I$; this is a consequence of Lemma G.1.

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

▶ **Lemma H.6.** *Suppose that we can derive $\Gamma \vdash^{\kappa}_{m} R[S/x] : \hat{\tau} \triangleright c$, where $ord(x) = m$. Then, for some finite set I, we can derive $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and $\Upsilon^x \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$, 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. 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 $\Upsilon^x \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$, but it actually equals the type judgment that we can derive by assumption.

In the sequel we assume that x is free in R. 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 $\Delta_1 \vdash^{\kappa}_{m} S : \hat{\sigma}_1 \triangleright d_1$ by assumption, and $\Upsilon^x \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$ using the (Var) rule.

Next, suppose that $R = \mathsf{nd}\langle P_1, \dots, P_r\rangle$. The original derivation ends with the (Nd) rule whose premiss is $\Gamma \vdash^{\kappa}_{m} P_k[S/x] : \hat{\tau} \triangleright c$ for some $k \in \{1, \dots, r\}$. 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 $\Upsilon^x \vdash^{\kappa}_{m} P_k : \hat{\tau} \triangleright e$

Next, suppose that $R = \lambda y.P$. We have $y \neq x$, and, as always during a substitution, we assume (by performing α-conversion) that y is not free in S. The original derivation ends with the (λ) rule, whose premiss is $\Gamma[y \mapsto C'] \vdash^{\kappa}_{m} P[S/x] : \hat{\tau}' \triangleright c$. We apply the induction assumption to this premiss, and we obtain a derivation of $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for

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

every $i \in I$, and of $\Upsilon^x[y \mapsto C''] \vdash^\kappa_m P : \hat{\tau}' \triangleright e$, 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 $\Upsilon^x[y \mapsto C'] \vdash^\kappa_m P : \hat{\tau}' \triangleright e$ we apply again the (λ) rule, which gives $\Upsilon^x \vdash^\kappa_m R : \hat{\tau} \triangleright e$.

Another possibility is that $R = a\langle P_1, \dots, P_r\rangle$, where $a \neq \mathsf{nd}$. Then the original derivation ends with the (Con≥1) rule (or with the (Con1) rule which is a special case of the (Con≥1) rule), whose premisses are $\Gamma_j \vdash^\kappa_m P_j[S/x] : \hat{\tau}_j \triangleright c_j$ for $j \in \{1, \dots, r\}$. We apply the induction assumption to these premisses. Assuming w.l.o.g. that the resulting sets I_j are disjoint, and taking $I = \bigcup_{j=1}^r I_j$, we obtain a derivation of $\Delta_i \vdash^\kappa_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and of $\Upsilon^x_j \vdash^\kappa_m P_j : \hat{\tau}_j \triangleright e_j$ for every $j \in \{1, \dots, r\}$, where, for every $j \in \{1, \dots, r\}$, 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 $c_j = e_j + \sum_{i\in I_j} d_i$. For $j \in \{1, \dots, r\}$ we have by Lemma G.1 that $\mathsf{Mk}(\Upsilon^x_j) \leq \mathsf{Mk}(\hat{\tau}_j)$, hence in particular $\sum_{i\in I_j} \mathsf{Mk}(\hat{\sigma}_i) \leq \mathsf{Mk}(\hat{\tau}_j)$. Since $\mathsf{Mk}(\hat{\tau}) = \sum_{j=1}^r \mathsf{Mk}(\hat{\tau}_j)$ (which follows from the (Con≥1) rule) we obtain that $\sum_{i\in I} \mathsf{Mk}(\hat{\sigma}_i) \leq \mathsf{Mk}(\hat{\tau})$. As $\mathsf{Mk}(\hat{\tau})$ is a marker multiset (i.e., contains every marker at most $|A|$ times), we can deduce that every unbalanced type triple appears as $\hat{\sigma}_i$ for at most $|A|$ 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. Another side condition of the (Con≥1) rule says that $(F, c) \in \mathit{Comp}_m(M; (\{(0, a)\}, \mathbf{0}), (F_1, c_1), \dots, (F_r, c_r))$ for appropriate arguments M, F, F_j. Taking $e = c + \sum_{j=1}^r (e_j - c_j)$ we also have that $(F, e) \in \mathit{Comp}_m(M; (\{(0, a)\}, \mathbf{0}), (F_1, e_1), \dots, (F_r, e_r))$. Having all this, we can apply the (Con≥1) rule again, deriving $\Upsilon^x \vdash^\kappa_m R : \hat{\tau} \triangleright e$. Simultaneously we observe that $c = e + \sum_{i\in I} d_i$.

Finally, suppose that $R = P\,Q$. This case is very similar to the previous one. The original derivation ends with the (@) rule, whose premisses are $\Gamma_0 \vdash^\kappa_m P[S/x] : \hat{\tau}_0 \triangleright c_0$ and $\Gamma_j \vdash^\kappa_m Q[S/x] : \hat{\tau}_j \triangleright c_j$ for $j \in J$, where we assume that $0 \notin J$. We apply the induction assumption to all these premisses. Assuming w.l.o.g. that the resulting sets I_j are disjoint, and taking $I = \bigcup_{j\in\{0\}\cup J} I_j$, we obtain a derivation of $\Delta_i \vdash^\kappa_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$, and of $\Upsilon^x_0 \vdash^\kappa_m P : \hat{\tau}_0 \triangleright e_0$, and of $\Upsilon^x_j \vdash^\kappa_m Q : \hat{\tau}_j \triangleright e_j$ 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 $c_j = e_j + \sum_{i\in I_j} d_i$. As previously, using Lemma G.1 we deduce that $\Upsilon^x = \Upsilon[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I|\!\}]$ is a type environment. By applying the (@) rule again, we derive $\Upsilon^x \vdash^\kappa_m R : \hat{\tau} \triangleright e$, where $e = c + \sum_{j\in\{0\}\cup J} (e_j - c_j)$. It holds that $c = e + \sum_{i\in I} d_i$. ◀

▶ **Lemma H.7.** *If $F \in \mathcal{F}^\kappa_m$, and $M \in \mathcal{M}^\kappa_m$, and $M(k) = 0$ for all $(k, a) \in F$ then $(F', c) \in \mathit{Comp}_m(M; (F, e), ((\emptyset, d_i))_{i\in I})$ if and only if $F' \subseteq F$ and $c = e + \sum_{i\in I} d_i$.*

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 $k = 0, \dots, m+1$ 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 $k = 0$ or $M(k-1) = 0$, we have $f'_{k,a} = 0$, while if $k > 0$ 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 $a \in A$ (since F is m-bounded). Finally, recall that $(F', c) \in \mathit{Comp}_m(M; (F, e), ((\emptyset, d_i))_{i\in I})$ 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 $a \in A$ (i.e., $c = e + \sum_{i\in I} d_i$). ◀

Proof of Lemma H.3. Recall that we are given a derivation of $\Gamma \vdash^\kappa_m Q : \hat{\tau} \triangleright c$, and a β-reduction $P \to_\beta Q$ that is of order m, and our goal is to derive $\Gamma \vdash^\kappa_m P : \hat{\tau} \triangleright c$.

Suppose first that $P = (\lambda x.R)\,S$ and $Q = R[S/x]$, where $\mathit{ord}(x) = m$. From Lemma H.6 we obtain a derivation of $\Delta_i \vdash^\kappa_m S : \hat{\sigma}_i \triangleright d_i$ for every $i \in I$ (for some set I), and a derivation of $\Upsilon[x \mapsto C] \vdash^\kappa_m R : \hat{\tau} \triangleright e$, 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 $\Upsilon[x \mapsto C] \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$ we apply the (λ) rule, deriving $\Upsilon \vdash^{\kappa}_{m} \lambda x.R : (F, M - \mathsf{Mk}(C), C{\to}\tau) \triangleright e$. Notice that $\mathsf{Mk}(C) \leq \mathsf{Mk}(\Upsilon[x \mapsto C]) \leq M$ by Lemma G.1, so it makes sense to use $M - \mathsf{Mk}(C)$. When $\kappa = \varepsilon$ and $\tau = C_1{\to}\dots{\to}C_s{\to}o$, we also need to know that $(M - \mathsf{Mk}(C) + \mathsf{Mk}(C) + \sum_{i=1}^{s} \mathsf{Mk}(C_i))(0) = 1$ (see 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 $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for $i \in I$, we want to apply the (@) rule. By 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 $\{\!|(F_i{\upharpoonright_{\leq ord(S)}}, M_i{\upharpoonright_{\leq ord(S)}}, \sigma_i) \mid i \in I|\!\}{\to}\tau$ that we have to derive for $\lambda x.R$ is indeed $C{\to}\tau$, and the side condition $ord(S) \leq m$ 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 $F_i{\upharpoonright_{>ord(S)}}$ are empty, and that $M(k) = 0$ for all $(k, a) \in F$ by definition of a type triple ($\hat{\tau} = (F, M, \tau)$ is a type triple), and hence $(F, c) \in Comp_m(M; (F, e), ((F_i{\upharpoonright_{>ord(S)}}, d_i))_{i\in I})$ by Lemma H.7. The condition $\{(k, a) \in F \mid M(k) = 0\} \subseteq F$ is clearly satisfied. Thus the (@) rule can be applied, and it derives $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$.

It remains to consider the general situation: the redex involved in the β-reduction $P \to_\beta Q$ 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 $U \to_\beta V$. In the derivation of $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright c$ 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 $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$. ◀

H.4 Proof of Lemma H.4

Recall that in Lemma H.4 we are given a type derivation of order m, and we want to convert it into a type derivation of order $m + 1$, without decreasing the flag counter too much. The order of the derivation can be raised without any problem, we only need to additionally place $|A|$ markers of order $m + 1$ in some leaves of the derivation. We notice, however, that in the original derivation the flag counter computed the number of order-$(m + 1)$ flags, while in the new derivation it computes the number of order-$(m + 2)$ flags. We thus have to ensure that many order-$(m + 2)$ flags are created in the new derivation. To this end, we appropriately choose where the order-$(m + 1)$ markers are placed. Let us now give more details.

For the rest of the subsection fix the order m. While thinking about the word variant of the type system, we assume that $m \geq 0$, and while thinking about the tree variant, we assume that $m \geq -1$. 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 (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: we first label the derivation tree by some additional flags and markers, and then basing on such a labeling we construct a derivation of order $m + 1$. Recall that A is the fixed set of symbols for which we solve the diagonal problem. For $B \subseteq A$, and for a derivation tree t, a *B-labeling of t* assigns some number of order-$(m + 1)$ markers to every leaf of t, and for every $a \in B$, some number of $(m + 1, a)$-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 $m+1$. A B-labeling ρ of t is *consistent*, if:

- for every node v of t having children $v_1,\dots,v_k$, and for every $a\in B$, ρ assigns at most $c_v(a)-c_{v_1}(a)-\dots-c_{v_k}(a)$ 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 $c_v(a)-c_{v_1}(a)-\dots-c_{v_k}(a)$ appearing above is always nonnegative: the flag counter in every node is not smaller than the sum of flag counters coming from the premisses.

▶ **Lemma H.8.** *Let t be a derivation tree with root r, and let $a\in A$. 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 $v_1,\dots,v_k$ with $k\geq 1$. Fix some s for which $c_{v_s}(a)$ is maximal, i.e., such that $c_{v_s}(a)\geq c_{v_i}(a)$ for all $i\in\{1,\dots,k\}$. 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 $i\in\{1,\dots,k\}\setminus\{s\}$ 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 $l=c_r(a)-c_{v_1}(a)-\dots-c_{v_k}(a)$, 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 $i\in\{1,\dots,k\}$, and thus $f=l=c_r(a)\geq\log_2 c_r(a)$. Suppose now that $c_{v_s}(a)>0$, and denote $l'=|\{i\in\{1,\dots,k\}\mid c_{v_i}(a)>0\}|$. 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 $i\in\{1,\dots,k\}$, we obtain

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

▶ **Lemma H.9.** *Let t be a derivation tree with root r. Then there exists a consistent A-labeling of t that assigns exactly $|A|$ markers and at least $\left\lfloor\frac{1}{|A|}\log_2 c_r(a)\right\rfloor$ a-flags, for every $a\in A$.*

Proof. We start by applying Lemma H.8 for every symbol $a\in A$, which results in a consistent a-labeling ρ_a of t. Basing 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 $|A|$ markers in total. For every node v of t such that ρ assigns some markers in t_v, and for every $a\in A$, 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 symbol $a\in A$. In a small step concerning some symbol 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 $a \in A$. Let f_a be the total number of a-flags assigned by ρ_a; by Lemma H.8 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 $|A|$ nodes $v \in V$. This means that during the first $\left\lfloor \frac{h_a}{|A|} \right\rfloor$ 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 will be assigned by ρ. In consequence, the number of a-flags assigned by ρ is at least $g_a + \left\lfloor \frac{h_a}{|A|} \right\rfloor \geq \left\lfloor \frac{f_a}{|A|} \right\rfloor \geq \left\lfloor \frac{1}{|A|} \log_2 c_r(a) \right\rfloor$. ◀

Next, we show how to raise the order of a type derivation basing on a consistent labeling. In this part, it is convenient to assume that the labeling is maximal, in the following sense: a consistent A-labeling ρ of a derivation tree t is called *maximal* if for every $a \in A$ and for every node v of t having children $v_1, \dots, v_k$, if ρ assigns some marker in t_v, then ρ assigns exactly $c_v(a) - c_{v_1}(a) - \dots - c_{v_k}(a)$ 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 $\mathit{New}_{\mathsf{M}}$ and $\mathit{New}_{\mathsf{Fc}}$: we say that $\mathit{New}_{\mathsf{M}}(M, \mu) = M'$ and $\mathit{New}_{\mathsf{Fc}}(F, \mu, f) = (F', c')$ if

- $M'(k) = M(k)$ for $k \neq m+1$, and $M'(m+1) = \mu$,
- if $\mu > 0$, then $F' = F$ and $c' = f$, and
- if $\mu = 0$, then $F' = F \cup \{(m+1, a) \mid f(a) > 0\}$ 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 $a \in A$, then in the new derivation that we construct, we will 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 H.10.** *Suppose that $(F, c) \in \mathit{Comp}_m(M; ((F_i, c_i))_{i \in I})$, where $F \in \mathcal{F}^{\kappa}_m$, and $m \geq 0$ if $\kappa = \varepsilon$, and $m \geq -1$ if $\kappa =\Delta$. For $i \in I$ let $\mu_i \in \mathbb{N}$ and $f_i \colon A \to \mathbb{N}$. 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 $a \in A$, and that either*

- *$F_i \in \mathcal{F}^{\kappa}_m$, or*
- *$f_i = 0$ and $F_i = \{(0, a)\}$ for some $a \in \Sigma$.*

In such a situation, $\mathit{New}_{\mathsf{Fc}}(F, \mu, f) \in \mathit{Comp}_{m+1}(\mathit{New}_{\mathsf{M}}(M, \mu); (\mathit{New}_{\mathsf{Fc}}(F_i, \mu_i, f_i))_{i \in I})$.

Proof. Denote $M' = \mathit{New}_{\mathsf{M}}(M, \mu)$, $(F', c') = \mathit{New}_{\mathsf{Fc}}(F, \mu, f)$, and $(F'_i, c'_i) = \mathit{New}_{\mathsf{Fc}}(F_i, \mu_i, f_i)$ for $i \in I$. We consider the numbers $f_{k,a}$ and $f'_{k,a}$ appearing in the definition of the predicate $\mathit{Comp}_m(M; ((F_i, c_i))_{i \in I})$. We also consider analogous numbers defined by the predicate

$Comp_{m+1}(M';((F'_i,c'_i))_{i\in I})$, and we call them $g_{k,a}$ and $g'_{k,a}$. Since $M'\restriction_{\leq m} = M\restriction_{\leq m}$ and $F'_i\restriction_{\leq m} = F_i\restriction_{\leq m}$ for $i \in I$, for every $a \in A$ we clearly have that $g_{k,a} = f_{k,a}$ for $k \leq m$, and $g'_{k,a} = f'_{k,a}$ for $k \leq m+1$. Moreover $g_{m+2,a} = g'_{m+2,a} + \sum_{i\in I} |F'_i \cap \{(m+2,a)\}| = g'_{m+2,a}$ since the sets F'_i are $(m+1)$-bounded (notice that $F_i = \{(0,a)\}$ need not to be m-bounded, but surely is $(m+1)$-bounded).

Let us now see that for all $i \in I$ and $a \in A$ it holds that

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

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 $(m+1,a) \notin F_i$ (since $F_i \in \mathcal{F}^\kappa_m$), and F'_i contains $(m+1,a)$ (by definition of F'_i), and $f_i(a) \leq 1$ (by assumption), which gives (1).

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

$$\begin{aligned}\sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) &= \sum_{i\in I} f_i(a) + f_{m+1,a}\\ &= f'_{m+1,a} + \sum_{i\in I}|F_i \cap \{(m+1,a)\}| + \sum_{i\in I} f_i(a)\\ &= g'_{m+1,a} + \sum_{i\in I}|F'_i \cap \{(m+1,a)\}| + \sum_{i\in I} c'_i(a)\\ &= g_{m+1,a} + \sum_{i\in I} c'_i(a)\,. \end{aligned}\tag{2}$$

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 $c'(a) = g_{m+2,a} + \sum_{i\in I} c'_i(a)$ for all $a \in A$. 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-bounded, and since $g_{k,a} = f_{k,a}$ for $k \leq m$. When $\mu > 0$, we have $F' = F$, and we are done. Suppose thus that $\mu = 0$. Then F' contains also elements $(m+1,a)$ for all $a \in A$ such that $f(a) > 0$. Concentrate on one such a. By assumption and by (2) we obtain

$$0 < f(a) \leq \sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) = g_{m+1,a} + \sum_{i\in I} c'_i(a)\,.$$

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 $g_{m+1,a} > 0$. We thus have $(m+1,a) \in \{(k,a) \mid g_{k,a} > 0\}$, as required.

Next, we fix some $a \in A$, and we prove that $c'(a) = g'_{m+2,a} + \sum_{i\in I} c'_i(a)$, which is what we need since $g_{m+2,a} = g'_{m+2,a}$. 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 $M'(m+1) = \mu = 0$, and thus $g'_{m+2,a} = 0$, which gives the thesis. Next, suppose that $\mu > 0$. In such a case, using (2), we obtain that

$$c'(a) = f(a) = \sum_{i\in I} f_i(a) + c(a) - \sum_{i\in I} c_i(a) = g_{m+1,a} + \sum_{i\in I} c'_i(a) = g'_{m+2,a} + \sum_{i\in I} c'_i(a)\,.$$

◀

▶ **Lemma H.11.** *Let t be a derivation tree deriving $\Gamma \vdash^\kappa_m R : (F,M,\tau) \triangleright c$ (where $m \geq 0$ if $\kappa = \varepsilon$, and $m \geq -1$ if $\kappa =\triangle$) such that $ord(R) \leq m+1$, and $\Gamma(x) \neq \mathbf{0}$ only for variables x of order at most m. Let also ρ be a maximal consistent A-labeling of t, which assigns (in total) $\mu \leq |A|$ markers and $f(a)$ a-flags, for every $a \in A$. Then we can derive $\Gamma \vdash^\kappa_{m+1} R : (F',M',\tau) \triangleright c'$, 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 $(m+1)$-bounded type κ-triple. By assumption $\hat{\tau}$ is an m-bounded type κ-triple, so $M \in \mathcal{M}^\kappa_m$. Since M' differs from M only on order $m+1$, and $M'(m+1) = \mu \leq |A|$, we obtain that $M' \in \mathcal{M}^\kappa_{m+1}$ (if $\kappa = \varepsilon$, from $M(0) \leq 1$ we additionally deduce $M'(0) \leq 1$ since then $m+1 \geq 1$). We also have $F \in \mathcal{F}^\kappa_m \subseteq \mathcal{F}^\kappa_{m+1}$. The set F' differs from F only when $\mu = 0$, and then it additionally contains those pairs $(m+1, a)$ for which $f(a) > 0$. By consistency of ρ we know that if $\mu = 0$ (i.e., if ρ assigns no markers) then $\sum_{a\in A} f(a) \leq 1$. Thus $(m+1, a), (m+1, b) \in F'$ implies $a = b$, which establishes that $F' \in \mathcal{F}^\kappa_{m+1}$. We also need to know that $M'(k) = 0$ for all $(k, a) \in F'$. For $k \leq m$ this is the case because $M'\restriction_{\leq m} = M$ and $F'\restriction_{\leq m} = F$, and by definition of F' we have $(m+1, a) \in F'$ only when $M'(m+1) = \mu = 0$. If $\kappa = \varepsilon$, we additionally 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 then $M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1$ and $M'(0) = M(0)$ due to $m+1 \geq 1$.

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 $ord(x) \leq m$, 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 $\Gamma \vdash^\kappa_{m+1} R : \hat{\tau}' \triangleright c'$ (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, $\Gamma \vdash^\kappa_m P_i : \hat{\tau} \triangleright c$ 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 $\Gamma \vdash^\kappa_{m+1} P_i : \hat{\tau}' \triangleright c'$. Applying back the (Nd) rule we derive $\Gamma \vdash^\kappa_{m+1} P : \hat{\tau}' \triangleright c'$.

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, $\Gamma[x \mapsto C'] \vdash^\kappa_m P : (F, M_\lambda, \tau_\lambda) \triangleright c$, 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 $ord(R) \leq m+1$, we have $ord(P) \leq m+1$ and $ord(x) \leq m$, so assumptions of the lemma are satisfied for t_v; the induction assumption gives us a derivation of $\Gamma[x \mapsto C'] \vdash^\kappa_{m+1} P : (F', M'_\lambda, \tau_\lambda) \triangleright c'$, where $M'_\lambda = \mathit{New}_{\mathsf{M}}(M_\lambda, \mu)$. The triple container C is $ord(x)$-bounded, thus since $ord(x) \leq m$ we have $M'(m+1) = \mu = M'_\lambda(m+1) = M'_\lambda(m+1) - \mathsf{Mk}(C)(m+1)$, and hence $M' = M'_\lambda - \mathsf{Mk}(C)$. Thus after applying back the (λ) rule we obtain a derivation of $\Gamma \vdash^\kappa_{m+1} R : \hat{\tau}' \triangleright c'$.

Next, suppose that $R = a\langle P_1, \ldots, P_r\rangle$, where $a \neq \mathsf{nd}$ and $r \geq 0$. The root of t has exactly r children $v_1, \ldots, v_r$, labeled by $\Gamma_i \vdash^\kappa_m P_i : (F_i, M_i, o) \triangleright c_i$ for $i \in \{1, \ldots, r\}$. Take $I = \{0, 1, \ldots, r\}$, $\mu_0 = 0$, $f_0 = \mathbf{0}$, $F_0 = \{(0, a)\}$, $c_0 = \mathbf{0}$. For $i \in \{1, \ldots, r\}$ 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 $a \in A$. By the induction assumption, for every $i \in \{1, \ldots, r\}$ we can derive $\Gamma_i \vdash^\kappa_{m+1} P_i : (F'_i, M'_i, o) \triangleright c'_i$, where $M'_i = \mathit{New}_{\mathsf{M}}(M_i, \mu_i)$ and $(F'_i, c'_i) = \mathit{New}_{\mathsf{Fc}}(F_i, \mu_i, f_i)$. The rule used in the root of t (which is either (Con0), or (Con1), or (Con$\geq$1)) ensures that $\Gamma = \bigsqcup_{i=1}^{r} \Gamma_i$, and $\tau = o$, and $(F, c) \in \mathit{Comp}_m(M; (F_0, c_0), (F_1, c_1), \ldots, (F_r, c_r))$; if $r > 0$ we also have $M = \sum_{i=1}^{r} M_i$. We want to apply Lemma H.10; 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 $a \in A$. 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 $\mathit{New}_{\mathsf{Fc}}(F_0, \mu_0, f_0) = (F_0, \mathbf{0})$. Thus by Lemma H.10 we obtain that $(F', c') \in \mathit{Comp}_{m+1}(M'; (F_0, \mathbf{0}), (F'_1, c'_1), \ldots, (F'_r, c'_r))$. Applying back the appropriate rule (namely

(Con0) if $r = 0$, and (Con1) if $r > 0$ and $\kappa = \varepsilon$, and (Con≥1) if $r > 0$ and $\kappa = \triangle$) we can derive $\Gamma \vdash^{\kappa}_{m+1} R : \hat{\tau}' \triangleright c'$.

Finally, suppose that $R = P\,Q$. Let $\Gamma_0 \vdash^{\kappa}_{m} P : (F_0, M_0, C \to \tau) \triangleright c_0$ and $\Gamma_i \vdash^{\kappa}_{m} Q : (F_i, M_i, \tau_i) \triangleright c_i$ 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 w.l.o.g. 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 $a \in A$. The (@) rule ensures that $\Gamma = \bigsqcup_{i \in \{0\} \cup I} \Gamma_i$ and $ord(Q) \leq m$, and by homogeneity of the sort of P we obtain that $ord(P) \leq ord(Q) + 1 \leq m + 1$. This allows us to apply the induction assumption, which gives us derivations of $\Gamma_0 \vdash^{\kappa}_{m+1} P : (F'_0, M'_0, C \to \tau) \triangleright c'_0$ and $\Gamma_i \vdash^{\kappa}_{m+1} Q : (F'_i, M'_i, \tau_i) \triangleright c'_i$ 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 $F'_i{\restriction}_{\leq m} = F_i{\restriction}_{\leq m}$ and $M'_i{\restriction}_{\leq m} = M_i{\restriction}_{\leq m}$ for all $i \in I$, and $ord(Q) \leq m$, we have that $C = \{\!|(F'_i{\restriction}_{\leq ord(Q)}, M'_i{\restriction}_{\leq ord(Q)}, \tau_i) \mid i \in I|\!\}$.
- From the original use of the (@) rule we know that $M = \sum_{i \in \{0\} \cup I} M_i$, and $(F, c) \in Comp_m(M; (F_0, c_0), ((F_i{\restriction}_{>ord(Q)}, c_i))_{i \in I})$. Assumptions of Lemma H.10 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 $a \in A$, 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 $m + 1$, and $ord(Q) \leq m$, we have $New_{\mathsf{Fc}}(F_i{\restriction}_{>ord(Q)}, \mu_i, f_i) = (F'_i{\restriction}_{>ord(Q)}, c'_i)$. Thus by Lemma H.10 we obtain that $(F', c') \in Comp_{m+1}(M'; (F'_0, c'_0), ((F'_i{\restriction}_{>ord(Q)}, c'_i))_{i \in I})$.
- 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 $m + 1$, we only need to check for all $a \in A$ that if $M'(m + 1) = 0$ and $(m + 1, a) \in F'_0$ then also $(m + 1, a) \in F'$. This is clear: $(m + 1, a) \in F'_0$ may only happen when $f_0(a) > 0$, but then $f(a) \geq f_0(a) > 0$, which in the case of $\mu = M'(m + 1) = 0$ implies that $(m + 1, a) \in F'$.

All this allows us to apply back the (@) rule, and derive $\Gamma \vdash^{\kappa}_{m+1} R : \hat{\tau}' \triangleright c'$. ◀

Proof of Lemma H.4. Consider a derivation tree t that derives $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$. Using Lemma H.9 we construct a consistent A-labeling ρ of t that assigns exactly $|A|$ markers and at least $\left\lfloor \frac{1}{|A|} \log_2 c(a) \right\rfloor$ a-flags, for every $a \in A$. W.l.o.g. we can assume that ρ is maximal: if not, we simply add more flags in some nodes, as required by the maximality condition.[7] Then Lemma H.11 gives us a derivation of $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m+1} \triangleright c'$ for some c' such that $c'(a) \geq \left\lfloor \frac{1}{|A|} \log_2 c(a) \right\rfloor$ for all $a \in A$, as required. ◀

I Soundness

In this section we prove the right-to-left implication of Theorems 3 and D.1. As a side effect, we also obtain a proof of Lemma G.4. We, basically, need to reverse the proof from the previous section. We give the following three lemmata.

▶ **Lemma I.1.** *Suppose that $P \to_{\beta(m)} Q$, where $m \geq 0$. If we can derive $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$, then we can also derive $\Gamma' \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$. Moreover, if the original derivation was wild, then the resulting one is also wild.*

[7] We notice that usually the labeling constructed by Lemma H.9 is not maximal.

▶ **Lemma I.2.** *Let P be a λ-term of complexity at most m. If we can derive $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$, where either $\kappa = \triangle$ or $m-1 \geq 0$, then we can also derive $\varepsilon \vdash^{\kappa}_{m-1} P : \hat{\rho}^{\kappa}_{m-1} \triangleright c'$ for some $c' \geq c$. Moreover, if the original derivation was wild, then the resulting one is also wild.*

▶ **Lemma I.3.** *Suppose that we can derive $\varepsilon \vdash_{0} P : \hat{\rho}_{0} \triangleright c$ or $\varepsilon \vdash^{\triangle}_{-1} P : \hat{\rho}^{\triangle}_{-1} \triangleright c$, where P is a λ-term of complexity 0. Then there exists a tree $T \in \mathcal{L}(P)$ such that for every $a \in A$, the number of appearances of a in T is $c(a)$.*

Let us now see how the right-to-left implication of Theorem 3 follows from these lemmata. Thus take a closed λ-term P of sort o and complexity at most $m+1$, and suppose that for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_{m} P : \hat{\rho}_{m} \triangleright c_{m}$ 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 symbol $a \in A$ appears at least n times. To this end, take some $n \in \mathbb{N}$, and the type judgment corresponding to this n. Using Lemma F.1 we can find a finite λ-term $P' \preccurlyeq P$ for which we can also derive $\varepsilon \vdash_{m} P' : \hat{\rho}_{m} \triangleright c_{m}$. Then we apply Lemma H.1 to P', obtaining λ-terms $Q_{m+1}, Q_{m}, \dots, Q_{0}$ such that, for every $k \in \{0, \dots, m\}$, 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 $k = m, m-1, \dots, 0$ we perform two steps. First, we repeatedly apply Lemma I.1 to every β-reduction (of order k) between Q_{k+1} and Q_{k}, obtaining a derivation of $\varepsilon \vdash_{k} Q_{k} : \hat{\rho}_{k} \triangleright c_{k}$. Then, if $k \geq 1$, we apply Lemma I.2, obtaining a derivation of $\varepsilon \vdash_{k-1} Q_{k} : \hat{\rho}_{k-1} \triangleright c_{k-1}$ for some $c_{k-1} \geq c$. We end up with a derivation of $\varepsilon \vdash_{0} Q_{0} : \hat{\rho}_{0} \triangleright c_{0}$, where $c_{0}(a) \geq c_{m}(a) \geq n$ for all $a \in A$. By Lemma I.3 we can find a tree $T \in \mathcal{L}(Q_{0}) = \mathcal{L}(BT(P'))$ such that for every $a \in A$, the number of appearances of a in T as at least n. Due to Lemma F.2, we also have $T \in \mathcal{L}(BT(P))$, as needed.

Similarly we prove the right-to-left implication of Theorem D.1. The only difference is that when we end up with a the type judgment $\varepsilon \vdash^{\triangle}_{m} Q_{0} : \hat{\rho}^{\triangle}_{0} \triangleright c_{0}$, we cannot directly apply Lemma I.3. But this time we should just apply Lemma I.2 one more time, and apply Lemma I.3 to the type judgment $\varepsilon \vdash^{\triangle}_{-1} Q_{0} : \hat{\rho}^{\triangle}_{-1} \triangleright c_{-1}$.

We also obtain a proof of Lemma G.4. Indeed, suppose that we have a wild derivation of $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c_{m}$ for a closed λ-term P of sort o and complexity at most $m+1$. Then, by applying the same arguments as above, we obtain a derivation of $\varepsilon \vdash^{\kappa}_{0} Q_{0} : \hat{\rho}^{\kappa}_{0} \triangleright c_{0}$ for some λ-term Q_{0} of complexity 0. Moreover, this derivation is wild, since Lemmata I.1 and I.2 preserve wildness. On the other hand, a λ-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 $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c_{m}$.

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

I.1 Proof of Lemma I.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 S. 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 appearance of x in the derivation concerning R. Unbalanced 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 G.3.

We start the proof by showing in Lemma I.4 how type derivations may be composed during a substitution. This lemma can be seen as a converse of Lemma H.6.

▶ **Lemma I.4.** *Suppose that we can derive $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for $i \in I$, and $\Upsilon^x \vdash^{\kappa}_{m} R : \hat{\tau} \triangleright e$, 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 $\Gamma' \vdash^{\kappa}_{m} R[S/x] : \hat{\tau} \triangleright c$ for $c = e + \sum_{i\in I} d_i$ and for some $\Gamma' \sqsubseteq \Gamma$. Moreover, if some of the original derivations is wild, then the resulting derivation is also wild.*

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

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 $\Gamma \vdash^{\kappa}_{m} R[S/x] : \hat{\tau} \triangleright c$ by assumption.

In the sequel we assume that x is free in R. 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 $\mathit{ord}(x) = m$, 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 $\Gamma \vdash^{\kappa}_{m} R[S/x] : \hat{\tau} \triangleright c$ 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 G.1 we obtain that $\mathsf{Mk}(\Delta_i) \leq \mathsf{Mk}(\hat{\sigma}_i) = \mathbf{0}$ for all $i \in I$, i.e., 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 G.3 we obtain that $d_i = \mathbf{0}$ for all $i \in I$, so $c = d_i$. Thus as the resulting derivation we can take $\Delta_i \vdash^{\kappa}_{m} S : \hat{\sigma}_i \triangleright d_i$ for any $i \in I$.

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

Next, suppose that $R = \lambda y.P$. We have $y \neq x$, and, as always during a substitution, we assume (by performing α-conversion) that y is not free in S. The derivation for R ends with the (λ) rule, whose premiss is $\Upsilon^x[y \mapsto C'] \vdash^{\kappa}_{m} P : \hat{\tau}' \triangleright e$. While writing $\Upsilon^x[y \mapsto C']$ we mean that $\Upsilon^x(y) = \mathbf{0}$, and since y is 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 $\Gamma'[y \mapsto C''] \vdash^{\kappa}_{m} P[S/x] : \hat{\tau}' \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$ and some $C'' \sqsubseteq C'$. We then apply again the (λ) rule obtaining $\Gamma' \vdash^{\kappa}_{m} R[S/x] : \hat{\tau} \triangleright c$, as needed.

Another possibility is that $R = a\langle P_1, \dots, P_r\rangle$, where $a \neq \mathsf{nd}$. Then the derivation for R ends with one of the rules (Con1) or (Con≥1), whose premisses are $\Upsilon^x_j \vdash^{\kappa}_{m} P_j : \hat{\tau}_j \triangleright e_j$ for $j \in \{1, \dots, r\}$. It holds that $\Upsilon[x \mapsto \{\!\!\{\hat{\sigma}_i \mid i \in I\}\!\!\}] = \Upsilon^x = \Upsilon^x_1 \sqcup \dots \sqcup \Upsilon^x_r$. Let us see that we can find sets $I_1, \dots, I_r$ such that $I = I_1 \cup \dots \cup I_r$ and $\{\!\!\{\hat{\sigma}_i \mid i \in I_j\}\!\!\} = \Upsilon^x_j(x)$ for all $j \in \{1, \dots, r\}$. 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 $j \in \{1, \dots, r\}$ 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 $\Upsilon^x(x)(\hat{\sigma}) = \sum_{j=1}^{r} \Upsilon^x_j(x)(\hat{\sigma})$, so we can split these elements i into sets $I_1, \dots, I_r$ so that exactly $\Upsilon^x_j(x)(\hat{\sigma})$ of them are taken to I_j (for $j \in \{1, \dots, r\}$).

Having these sets, for every $j \in \{1, \dots, r\}$ 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 G.1 we obtain that all type triples in Δ_i are balanced, and thus $\Delta_i \sqcup \Delta_i = \Delta_i$, and from Lemma G.3 we obtain that $d_i = \mathbf{0}$. These lemmata can be used, since $\mathit{ord}(S) = \mathit{ord}(x) = m \leq m$. 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 $\bigsqcup_{i\in I} \Delta_i = \bigsqcup_{j=1}^{r} \bigsqcup_{i\in I_j} \Delta_i$ and $\sum_{i\in I} d_i = \sum_{j=1}^{r} \sum_{i\in I_j} d_i$. 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 $j \in \{1,\dots,r\}$, we have that $\Gamma = \Gamma_1 \sqcup \dots \sqcup \Gamma_r$ and $c = c_1 + \dots + c_r$. In particular $\Gamma_j(y) \leq \Gamma(y)$ for every $j \in \{1,\dots,r\}$ and every variable y, so Γ_j is a type environment (i.e., $\Gamma_j(x)$ contains every unbalanced type triple at most $|A|$ times).

Now for every $j \in \{1,\dots,r\}$ we apply the induction assumption to the premiss $\Upsilon_j^x \vdash_m^\kappa P_j : \hat{\tau}_j \triangleright e_j$ and to type judgments $\Delta_i \vdash_m^\kappa S : \hat{\sigma}_i \triangleright d_i$ only for $i \in I_j$; we obtain a derivation of $\Gamma'_j \vdash_m^\kappa P_j[S/x] : \hat{\tau}_j \triangleright c_j$ for $c_j = e_j + \sum_{i\in I_j} d_i$ and for some $\Gamma'_j \sqsubseteq \Gamma_j$. To the obtained type judgments we apply the appropriate rule, (Con1), or (Con≥1), and we derive $\Gamma' \vdash_m^\kappa R[S/x] : \hat{\tau} \triangleright c$ for $\Gamma' = \Gamma'_1 \sqcup \dots \sqcup \Gamma'_r$. We need to notice here that $c - e = \sum_{j=1}^r (e_j - c_j)$, and thus if $(F,e) \in \mathit{Comp}_m(M; (\{(0,a)\}, \mathbf{0}), (F_1,c_1), \dots, (F_r,c_r))$ for some arguments M, F, F_j (as ensured by the original use of the rule), then also $(F,c) \in \mathit{Comp}_m(M; (\{(0,a)\}, \mathbf{0}), (F_1,c_1), \dots, (F_r,c_r))$ (as needed for the new use of the rule). We also notice that $\Gamma' \sqsubseteq \Gamma$.

Finally, suppose that $R = P\,Q$. This case is very similar to the previous one. The derivation for R ends with the (@) rule, whose premisses are $\Upsilon_0^x \vdash_m^\kappa P : \hat{\tau}_0 \triangleright e_0$ and $\Upsilon_j^x \vdash_m^\kappa Q : \hat{\tau}_j \triangleright e_j$ for $j \in J$, where we assume that $0 \notin J$. As in the previous case we can find sets $(I_j)_{j\in\{0\}\cup J}$ such that $I = \bigcup_{j\in\{0\}\cup J} I_j$ and $\{\!|\hat{\sigma}_i \mid i \in I_j|\!\} = \Upsilon_j^x(x)$ for all $j \in \{0\} \cup J$. Again we write $\Upsilon_j^x = \Upsilon_j[x \mapsto \{\!|\hat{\sigma}_i \mid i \in I_j|\!\}]$, and $\Gamma_j = \Upsilon_j \sqcup \bigsqcup_{i\in I_j} \Delta_i$, and $c_j = e_j + \sum_{i\in I_j} d_i$ for $j \in \{0\} \cup J$, and we have that $\Gamma = \bigsqcup_{j\in\{0\}\cup J} \Gamma_j$ and $c = \sum_{j\in\{0\}\cup J} c_j$. We then apply the induction assumption to all premisses, and we obtain derivations of $\Gamma'_0 \vdash_m^\kappa P[S/x] : \hat{\tau}_0 \triangleright c_0$ and of $\Gamma'_j \vdash_m^\kappa Q[S/x] : \hat{\tau}_j \triangleright c_j$ for $j \in J$, where $\Gamma'_j \sqsubseteq \Gamma_j$ for $j \in \{0\} \cup J$. By applying the (@) rule again, we derive $\Gamma' \vdash_m^\kappa R[S/x] : \hat{\tau} \triangleright c$ for $\Gamma' = \bigsqcup_{j\in\{0\}\cup J} \Gamma'_j \sqsubseteq \Gamma$.

We also need to see that if some of the original derivation is wild, then the resulting derivation is wild as well. Indeed, suppose that in the derivation of $\Upsilon^x \vdash_m^\kappa R : \hat{\tau} \triangleright e$ 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 λ-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 $\Delta_i \vdash_m^\kappa S : \hat{\sigma}_i \triangleright d_i$ for some $i \in I$. By Lemma G.5 this can happen only when $\hat{\sigma}_i$ is unbalanced (recall that $\mathit{ord}(S) = m$, 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 I.1. Recall that we are given a derivation of $\Gamma \vdash_m^\kappa P : \hat{\tau} \triangleright c$, and a β-reduction $P \to_\beta Q$ that is of order m, and our goal is to derive $\Gamma' \vdash_m^\kappa Q : \hat{\tau} \triangleright c$ for some $\Gamma' \sqsubseteq \Gamma$.

Suppose first that $P = (\lambda x.R)\,S$ and $Q = R[S/x]$, where $\mathit{ord}(x) = m$. Then the given derivation ends with the (@) rule, whose premisses are $\Upsilon \vdash_m^\kappa \lambda x.R : \hat{\tau}_\lambda \triangleright e$ and $\Delta_i \vdash_m^\kappa S : \hat{\sigma}_i \triangleright d_i$ 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 $\Upsilon^x \vdash_m^\kappa R : (F', M'', \tau) \triangleright e$, 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 $\mathit{ord}(S) = \mathit{ord}(x) = m$, for all $i \in I$ we have that $F_i{\restriction_{\leq \mathit{ord}(S)}} = F_i$, and $M_i{\restriction_{\leq \mathit{ord}(S)}} = M_i$, and $F_i{\restriction_{> \mathit{ord}(S)}} = \emptyset$. Conditions of the (@) rule imply that:

1. $\Gamma = \Upsilon \sqcup \bigsqcup_{i\in I} \Delta_i$;
2. $C = \{\!|(F_i{\restriction_{\leq \mathit{ord}(S)}}, M_i{\restriction_{\leq \mathit{ord}(S)}}, \sigma_i) \mid i \in I|\!\} = \{\!|\hat{\sigma}_i \mid i \in I|\!\}$;
3. $M = M' + \sum_{i\in I} M_i = M'' - \mathsf{Mk}(C) + \sum_{i\in I} M_i = M''$;
4. $(F,c) \in \mathit{Comp}_m(M; (F',e), ((F_i{\restriction_{> \mathit{ord}(S)}}, d_i))_{i\in I})$, which by Lemma H.7 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);

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 G.1, and $d_i = \mathbf{0}$ by Lemma G.3. 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 I.4 to $\Upsilon^x \vdash^\kappa_m R : \hat{\tau} \triangleright e$ and to $\Delta_i \vdash^\kappa_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I'$; we obtain a derivation of $\Gamma' \vdash^\kappa_m Q : \hat{\tau} \triangleright c$ 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 $(k,a) \in F{\restriction}_{>ord(Q)} = \emptyset$. Moreover, the removed subderivations ending with $\Delta_i \vdash^\kappa_m S : \hat{\sigma}_i \triangleright d_i$ for $i \in I \setminus I'$ cannot be wild by Lemma G.5. Thus the wild use of the (@) rule is located in some of the subderivations passed to Lemma I.4, and thus it is preserved.

It remains to consider the general situation: the redex involved in the β-reduction $P \to_\beta Q$ 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 most rules, we only 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 $\Gamma[x \mapsto C'] \vdash^\kappa_m R : (F, M, \tau) \triangleright c$ and its conclusion is $\Gamma \vdash^\kappa_m \lambda x.R : (F, M - \mathsf{Mk}(C), C {\to} \tau) \triangleright c$, by the induction assumption we obtain a derivation of $\Gamma'[x \mapsto C''] \vdash^\kappa_m S : (F, M, \tau) \triangleright c$ with $\Gamma' \sqsubseteq \Gamma$ and $C'' \sqsubseteq C' \sqsubseteq C$; we can then apply the (λ) rule and derive $\Gamma' \vdash^\kappa_m S : (F, M - \mathsf{Mk}(C), C {\to} \tau) \triangleright c$. ◀

▶ Remark. Recall that the (λ) rule allows to forget about some balanced type triples provided by an argument, i.e., we can have $C' \sqsubseteq C$. We notice, however, that in derivations constructed in Section H we use the (λ) rule only for $C' = C$. This means that Theorem 3 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 I.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 it starts to hold $C' \sqsubseteq C$ instead of $C' = C$. Thus even if we start from a derivation in the more restrictive type system (with $C' = C$), in the soundness proof we pass through derivations in the original type system (with $C' \sqsubseteq C$).

I.2 Proof of Lemma I.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, the 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 behaviour of the $Comp_m$ predicate.

▶ **Lemma I.5.** *Suppose that $(F, c) \in Comp_m(M; ((F_i, c_i))_{i\in I})$, and $m \geq 0$, and that $M(k) = 0$ for all $(k,a) \in F$. Suppose also that for every $i \in I$ either*

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

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

Proof. We consider the numbers $f_{k,a}$ and $f'_{k,a}$ appearing in the definition of the predicate $\mathit{Comp}_m(M; ((F_i, c_i))_{i\in I})$. We also consider analogous numbers defined by the predicate $\mathit{Comp}_{m-1}(M{\restriction_{\leq m-1}}; ((F'_i, c'_i))_{i\in I})$, and we call them $g_{k,a}$ and $g'_{k,a}$. Since the arguments are the same up to order $m-1$, for every $a \in A$ 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 $a \in A$, 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 $a \in A$. For the rest of the proof fix some $a \in A$. 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
- $(F_i, c_i) = (F'_i, c'_i) = (\{(0, b)\}, \mathbf{0})$ 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

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

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 ($m \geq 0$), it holds 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:

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

We now generalize Lemma I.2 to arbitrary type judgments.

▶ **Lemma I.6.** *Let P be a λ-term of complexity at most m, whose all free variables are of order at most $m-1$. If we can derive $\Gamma \vdash^{\kappa}_{m} P : (F, M, \tau) \triangleright c$, where either $\kappa = \vartriangle$ or $m - 1 \geq 0$, then we can also derive $\Gamma \vdash^{\kappa}_{m-1} P : (F{\restriction_{\leq m-1}}, M{\restriction_{\leq m-1}}, \tau) \triangleright c'$ for some $c' \colon A \to \mathbb{N}$ such that $c'(a) \geq c(a) + |F \cap \{(m, a)\}|$ for all $a \in A$. 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 $\Gamma \vdash^{\kappa}_{m} P : \hat{\tau} \triangleright c$. 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 $\Gamma = \varepsilon[x \mapsto (F, M{\restriction_{\leq \mathit{ord}(x)}}, \tau)]$, and $c = \mathbf{0}$. By assumption of the lemma we have $\mathit{ord}(x) \leq m - 1$, so $(M{\restriction_{\leq m-1}}){\restriction_{\leq \mathit{ord}(x)}} = M{\restriction_{\leq \mathit{ord}(x)}}$ and $F{\restriction_{\leq m-1}} = F$ (because F is $\mathit{ord}(x)$-bounded). In consequence, we can use the (Var) rule to derive $\Gamma \vdash^{\kappa}_{m-1} P : \hat{\sigma} \triangleright \mathbf{0}$.

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

Next, suppose that $P = \lambda x.Q$. Then the final (λ) rule has a premiss $\Gamma[x \mapsto C'] \vdash^{\kappa}_{m} Q : (F, M', \tau') \triangleright c$, 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 $\Gamma[x \mapsto C'] \vdash^{\kappa}_{m-1} Q : (F\restriction_{\leq m-1}, M'\restriction_{\leq m-1}, \tau') \triangleright c'$ with c' such that $c'(a) \geq c(a) + |F \cap \{(m, a)\}|$ for all $a \in A$. 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 $\Gamma \vdash^{\kappa}_{m-1} P : \hat{\sigma} \triangleright c'$.

Next, suppose that $P = b\langle P_1, \dots, P_r\rangle$ with $b \neq \mathsf{nd}$. Then $\tau = o$, and the final rule (being either (Con0), or (Con1), or (Con$\geq$1)) has premisses $\Gamma_i \vdash^{\kappa}_{m} P_i : (F_i, M_i, o) \triangleright c_i$ for $i \in \{1, \dots, r\}$. By the induction assumption, for every $i \in \{1, \dots, r\}$ we can derive $\Gamma_i \vdash^{\kappa}_{m-1} P_i : (F_i\restriction_{\leq m-1}, M_i\restriction_{\leq m-1}, o) \triangleright c'_i$ for some c'_i such that $c'_i(a) \geq c_i(a) + |F_i \cap \{(m, a)\}|$ for all $a \in A$. If $r > 0$, we have a side condition $M = M_1 + \dots + M_r$, which implies $M\restriction_{\leq m-1} = M_1\restriction_{\leq m-1} + \dots + M_r\restriction_{\leq m-1}$. Another side condition says that $(F, c) \in \mathit{Comp}_m(M; (\{(0, b)\}, \mathbf{0}), (F_i, c_i)_{i \in \{1,\dots,r\}})$, and we need to see that $(F\restriction_{\leq m-1}, c') \in \mathit{Comp}_{m-1}(M\restriction_{\leq m-1}; (\{(0, b)\}, \mathbf{0}), (F_i\restriction_{\leq m-1}, c'_i)_{i \in \{1,\dots,r\}})$ for some c' such that $c'(a) \geq c(a) + |F \cap \{(m, a)\}|$ for all $a \in A$; this follows directly from Lemma I.5. Thus we can apply back the appropriate rule, and derive $\Gamma \vdash^{\kappa}_{m-1} P : \hat{\sigma} \triangleright c'$.

Finally, suppose that $P = Q\,R$. Then the final (@) rule has premisses $\Gamma' \vdash^{\kappa}_{m} Q : (F', M', C \to \tau) \triangleright e$ and $\Gamma_i \vdash^{\kappa}_{m} R : (F_i, M_i, \tau_i) \triangleright d_i$ 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 $\Gamma' \vdash^{\kappa}_{m-1} Q : (F'\restriction_{\leq m-1}, M'\restriction_{\leq m-1}, C \to \tau) \triangleright e'$ with e' such that $e'(a) \geq e(a) + |F' \cap \{(m, a)\}|$ for all $a \in A$, and, for all $i \in I$, a derivation of $\Gamma_i \vdash^{\kappa}_{m-1} R : (F_i\restriction_{\leq m-1}, M_i\restriction_{\leq m-1}, \tau_i) \triangleright d'_i$ with d'_i such that $d'_i(a) \geq d_i(a) + |F_i \cap \{(m, a)\}|$ for all $a \in A$. The side condition $M = M' + \sum_{i \in I} M_i$ implies $M\restriction_{\leq m-1} = M'\restriction_{\leq m-1} + \sum_{i \in I} M_i\restriction_{\leq m-1}$. Another side condition says that $(F, c) \in \mathit{Comp}_m(M; (F', e), (F_i\restriction_{> ord(R)}, d_i)_{i \in I})$. 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 $a \in A$, thus by Lemma I.5 we obtain $(F\restriction_{\leq m-1}, c') \in \mathit{Comp}_{m-1}(M\restriction_{\leq m-1}; (F'\restriction_{\leq m-1}, e'), ((F_i\restriction_{> ord(R)})\restriction_{\leq m-1}, d'_i)_{i \in I})$ for some c' such that $c'(a) \geq c(a) + |F \cap \{(m, a)\}|$ for all $a \in A$. Trivially $(F_i\restriction_{> ord(R)})\restriction_{\leq m-1} = (F_i\restriction_{\leq m-1})\restriction_{> ord(R)}$. 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 $\Gamma \vdash^{\kappa}_{m-1} P : \hat{\sigma} \triangleright c'$.

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 $l \in \{k, k+1, \dots, m\}$. If $k = m$, the type judgment derived by the induction assumption, $\Gamma_i \vdash^{\kappa}_{m-1} R : (F_i\restriction_{\leq m-1}, M_i\restriction_{\leq m-1}, \tau_i) \triangleright d'_i$, 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 G.3 (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 $l \in \{k, k+1, \dots, m-1\}$. 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 contain some wild use of the (@) rule, the resulting derivation also contains a wild use of the (@) rule, in the same place. ◀

Lemma I.2 is obtained by specializing Lemma I.6 to the situation when P is closed, and

$(F, M, \tau) = \hat{\rho}^{\kappa}_{m}$. Notice that then $(F{\restriction_{\leq m-1}}, M{\restriction_{\leq m-1}}, \tau) = \hat{\rho}^{\kappa}_{m-1}$.

I.3 Proof of Lemma I.3

In this lemma, we are given a derivation of $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$, where either $\kappa = \varepsilon$ and $m = 0$, or $\kappa = \triangle$ and $m = -1$, and where P is of complexity 0. The proof is by induction on the structure of some fixed derivation of $\varepsilon \vdash^{\kappa}_{m} P : \hat{\rho}^{\kappa}_{m} \triangleright c$. 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 λ-binder, nor an application. Thus P starts with a node constructor, $P = b\langle P_1, \dots, P_r\rangle$. We have two cases.

An easier case is when $b = \mathsf{nd}$. Then the final (ND) rule has one premiss $\varepsilon \vdash^{\kappa}_{m} P_i : \hat{\rho}^{\kappa}_{m} \triangleright c$ for some $i \in \{1, \dots, r\}$. The induction assumption gives us a tree T such that $P_i \to^{*}_{\mathsf{nd}} T$ and for every $a \in A$, the number of appearances of a in T is $c(a)$. Since $P \to_{\mathsf{nd}} P_i$, this gives the thesis.

Suppose now that $b \neq \mathsf{nd}$. Recall that for $(F, M, \tau) \in \mathcal{TT}^{\triangle o}_{-1}$, the flag set F and the marker multiset M should be (-1)-bounded, hence $F = \emptyset$ and $M = \mathbf{0}$, and the type $\tau \in \mathcal{T}^{o}$ can only be o. For $(F, M, \tau) \in \mathcal{TT}^{o}_{0}$ we are required that $\tau = o$, that F and M are 0-bounded, that $M(0) = 1$ (hence $M = \{\!|0|\!\}$), and that $M(k) = 0$ for all $(k, a) \in F$ (hence $F = \emptyset$). In consequence $\hat{\rho}^{\kappa}_{m}$ is the only type triple in $\mathcal{TT}^{\kappa o}_{m}$. It follows that premisses of the final rule are of the form $\varepsilon \vdash^{\kappa}_{m} P_i : \hat{\rho}^{\kappa}_{m} \triangleright c_i$ for $i \in \{1, \dots, r\}$. By the induction assumption, for every $i \in \{1, \dots, r\}$ we obtain a tree T_i such that $P_i \to^{*}_{\mathsf{nd}} T_i$ and for every $a \in A$, the number of appearances of a in T_i is $c_i(a)$. We take $T = b\langle T_1, \dots, T_r\rangle$; then $P \to^{*}_{\mathsf{nd}} T$. As in the proof of Lemma H.2, we observe that $(\emptyset, c) \in \mathit{Comp}_m(\mathsf{Mk}(\hat{\rho}^{\kappa}_{m}); (\{(0, b)\}, \mathbf{0}), (\emptyset, c_1), \dots, (\emptyset, c_r))$ holds exactly when $c(a) = c_1(a) + \dots + c_r(a)$ for $a \in A \setminus \{b\}$, and $c(a) = 1 + c_1(a) + \dots + c_r(a)$ if $a = b \in A$. It follows that for every $a \in A$, the number of appearances of a in T is $c(a)$.

J Complexity

In this section we prove Theorems 2 and 5. In the first part we concentrate on the upper bound. We are thus given a set A and a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$ of order at most $m + 1$, and we want to decide whether $\mathit{Diag}_A(\mathcal{L}(\mathcal{G}))$ holds. This should be done:

- in $(m + 1)$-EXPTIME for $m \geq 0$, and in NP for $m = -1$, when $\mathcal{G}$ can be arbitrary, and
- in m-EXPTIME for $m \geq 1$, and in NP for $m = 0$, assuming that $\mathcal{G}$ is word-recognizing.

Set $\kappa = \triangle$ in the former case, and $\kappa = \varepsilon$ in the latter case. In the cases resulting in an NP algorithm (i.e., $\kappa = \triangle, m = -1$ or $\kappa = \varepsilon, m = 0$) we also prove that the problem is fixed-parameter tractable when $|A|$ is viewed as a parameter. Due to Theorems 3 and D.1, the problem boils down to checking whether for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash^{\kappa}_{m} \Lambda(\mathcal{G}) : \hat{\rho}^{\kappa}_{m} \triangleright c_n$ 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 $\mathcal{G}$).

Let us recall from page 11 three definitions.

- Two type judgments are *equivalent* if they differ only in values of the flag counter.
- A derivation is *pumpable* if for every symbol $a \in A$, there are two equivalent type judgments lying on one branch of the derivation and such that the a-coordinate of their flag counter differs.
- We say that a type judgment $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright d$ 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 are free in Q.

Let $\mathcal{U}^{\mathcal{G}}$ be the set of useful type judgments $\Gamma \vdash_m^\kappa Q : \hat{\tau} \triangleright d$ satisfying $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$,[8] 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 G.1 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 $\varepsilon \vdash_m^\kappa \Lambda(\mathcal{G}) : \hat{\rho}_m^\kappa \triangleright c$ contain only type judgments from $\mathcal{U}^{\mathcal{G}}$.

J.1 Number of Equivalence Classes

In the first part, we bound the size of $\mathcal{U}^{\mathcal{G}}_{/\sim}$, giving more details than in Section 3.

For $n \in \{0, \dots, m+1\}$, denote by η_n^κ the maximum of $|\mathcal{TT}_k^{\kappa\alpha}|$ over all $k \in \{-1, 0, \dots, n\}$ and over all sorts α such that $|\alpha| \leq 2 \cdot |\mathcal{G}|$ and $\mathit{ord}(\alpha) \leq n$. In order to bound η_n^κ, let us take this number k and this sort $\alpha = \alpha_1 \to \dots \to \alpha_s \to o$ for which the maximum is reached. A type triple in $\mathcal{TT}_k^{\kappa\alpha}$ contains a flag set F, a marker multiset M, and a type $C_1 \to \dots \to C_s \to o$. In the flag set, for every order in $l \in \{0, \dots, k\}$ we either have no flags of order l, or we have an (l, a)-flag for some $a \in A$ (and no other flags of order l); this gives $(|A|+1)^{k+1}$ possibilities. In the marker multiset, for every order in $l \in \{0, \dots, k\}$ the number of order-l markers is in $\{0, \dots, |A|\}$; this also gives $(|A|+1)^{k+1}$ possibilities. Every triple container C_i is a function from $\mathcal{TT}_{\mathit{ord}(\alpha_i)}^{\kappa\alpha_i}$ to $\{0, \dots, |A|\}$; this gives $(|A|+1)^{|\mathcal{TT}_{\mathit{ord}(\alpha_i)}^{\kappa\alpha_i}|}$ possibilities. Because $\mathit{ord}(\alpha_i) \leq \mathit{ord}(\alpha) - 1 \leq n - 1$ and $|\alpha_i| \leq |\alpha| \leq 2 \cdot |\mathcal{G}|$, we have $|\mathcal{TT}_{\mathit{ord}(\alpha_i)}^{\kappa\alpha_i}| \leq \eta_{n-1}^\kappa$. Since moreover $s \leq |\alpha| \leq 2 \cdot |\mathcal{G}|$ and $k \leq n \leq m+1$, for $n \geq 1$ we obtain:

$$\begin{aligned}\eta_n^\kappa = |\mathcal{TT}_k^{\kappa\alpha}| &\leq (|A|+1)^{k+1} \cdot (|A|+1)^{k+1} \cdot \prod_{i=1}^{s} (|A|+1)^{|\mathcal{TT}_{\mathit{ord}(\alpha_i)}^{\kappa\alpha_i}|} \\ &\leq (|A|+1)^{2m+4+2\cdot|\mathcal{G}|\cdot\eta_{n-1}^\kappa}\,.\end{aligned}$$

For $n = 0$ we necessarily have $s = 0$, and thus $\eta_0^\kappa \leq (|A|+1)^2$. It follows that for all $n \in \mathbb{N}$, η_n^κ is at most n-fold exponential in $|\mathcal{G}|$ and $|A|$ (where by "0-fold exponential" we mean "polynomial").

In the case of $\kappa = \varepsilon$ we need a stronger bound. It can be established, because for every type triple $(F, M, C_1 \to \dots \to C_s \to o) \in \mathcal{TT}_k^\alpha$ we have the additional requirement that $M(0) + \sum_{i=1}^s \mathsf{Mk}(C_i)(0) = 1$. For $\mathit{ord}(\alpha) = 0$, this condition implies that $M(0) = 1$ (since then $s = 0$). In consequence, for $\mathit{ord}(\alpha) = 1$ the triple containers $C_1, \dots, C_s$ can contain altogether only at most one type triple (as every type triple in some C_i adds one to the sum $\sum_{i=1}^s \mathsf{Mk}(C_i)(0)$). It follows that we only need to remember which one of $C_1, \dots, C_s$ is nonempty (or that all of them are empty), and which type triple from $\mathcal{TT}_0^o$ it contains.[9] Thus for k and α maximizing η_1^ε we obtain:

$$\eta_1^\varepsilon = |\mathcal{TT}_k^\alpha| \leq (|A|+1)^{k+1} \cdot (|A|+1)^{k+1} \cdot (1 + s \cdot |\mathcal{TT}_0^o|) \leq (|A|+1)^4 \cdot (1 + 2 \cdot |\mathcal{G}| \cdot \eta_0^\varepsilon)\,.$$

This means that η_n^ε for $n \geq 1$ is at most $(n-1)$-fold exponential in $|\mathcal{G}|$ and $|A|$.

Recall from the proof of Proposition C.1 that by $\Lambda_{\mathcal{G}}(P)$ we denote the λ-term obtained by recursively expanding all nonterminals in a λ-term P (which could contain nonterminals). 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. It is easy to see that every subterm of $\Lambda(\mathcal{G})$ equals $\Lambda_{\mathcal{G}}(P)$ for some

[8] As one can see in the proof, the inequality $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$ is important only when $\kappa = \varepsilon$ and $m = 0$.
[9] Actually, $\mathcal{TT}_0^o$ contains only one element, namely $\hat{\rho}_0 = (\emptyset, \{\!\{0\}\!\}, o)$, but this is irrelevant here.

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 us now prove that if P of sort α is a subterm of $\mathcal{R}(N)$ for some nonterminal $N \in \mathcal{N}$, then $|\alpha| \leq |P| + |\mathcal{G}|$ (so in particular $|\alpha| \leq 2 \cdot |\mathcal{G}|$).[10] This is 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 we have $|\gamma| \leq |Q|$. 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 $|\mathcal{G}|$.

We now bound the size of $\mathcal{U}^{\mathcal{G}}_{/\sim}$, that is, the number of equivalence classes of useful type judgments $\Gamma \vdash^{\kappa}_{m} Q : \hat{\tau} \triangleright d$ satisfying $\mathsf{Mk}(\Gamma) \leq \mathsf{Mk}(\hat{\tau})$. As already said, there are at most $|\mathcal{G}|$ choices for Q. The sort of $Q = \Lambda_{\mathcal{G}}(P)$ is the same as the sort of P, thus (by the above) the size of this sort is at most $2 \cdot |\mathcal{G}|$. The order of Q is at most $m+1$, thus there are at most η^{κ}_{m+1} choices for $\hat{\tau}$. We now bound the number of choices for the type environment Γ. Recall that we may take $\Gamma(x) \neq \mathbf{0}$ only for variables x that are free in Q. All such variables are subterms of $\Lambda(x)$, so there exist at most $|\mathcal{G}|$ of them. Moreover, every free variable x of Q 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 three cases, depending on m.

- Suppose first that $\kappa = \triangle$ and $m \geq 0$, or $\kappa = \varepsilon$ and $m \geq 1$. Consider a free variable x^{α} of Q. Its sort α satisfies $|\alpha| \leq 2 \cdot |\mathcal{G}|$, so we obtain $|\mathcal{TT}^{\kappa\alpha}_{ord(\alpha)}| \leq \eta^{\kappa}_{m}$. The type environment Γ assigns to x^{α} a triple container from $\mathcal{TC}^{\kappa\alpha}$, that is a function from $\mathcal{TT}^{\kappa\alpha}_{ord(\alpha)}$ to $\{0, \dots, |A|\}$. The number of such functions is at most $(|A|+1)^{\eta^{\kappa}_{m}}$. By taking a product over all free variables of Q, we obtain that the number of possible type environments Γ is at most $(|A|+1)^{|\mathcal{G}| \cdot \eta^{\kappa}_{m}}$. This number is at most $(m+1)$-fold exponential in $|\mathcal{G}|$ and $|A|$ for $\kappa = \triangle$, and at most m-fold exponential in $|\mathcal{G}|$ and $|A|$ for $\kappa = \varepsilon$.
- Suppose now that $\kappa = \varepsilon$ and $m = 0$ (then the bound from the previous item is exponential, while we need a polynomial one). In this case all free variables of Q are of order 0 (thus of sort o). As already noticed, we have $\mathsf{Mk}(\hat{\sigma})(0) = 1$ for all $\hat{\sigma} \in \mathcal{TT}^{o}_{0}$. On the other hand $\mathsf{Mk}(\hat{\tau})(0) \leq 1$ (by 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, to at most one variable. We thus only need to remember which type triple it is, and to which variable it is assigned. We have at most $1 + |\mathcal{G}| \cdot \eta^{\varepsilon}_{0}$ possibilities (thus polynomially many).
- Finally, suppose that $\kappa = \triangle$ and $m = -1$. In this case Q has no free variables, so necessarily $\Gamma = \varepsilon$.

Altogether, it follows that $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$ is at most $(m+1)$-fold exponential in $|\mathcal{G}|$ and $|A|$ for $\kappa = \triangle$, and at most m-fold exponential in $|\mathcal{G}|$ and $|A|$ for $\kappa = \varepsilon$.

J.2 Pumpable Derivations

In the second subsection, we argue that we are actually interested in finding a pumpable derivation. We shall see here derivations as trees, similarly as in Appendix H.4. For a node v of a derivation tree, by c_v we denote the flag counter being part of the type judgment

[10] 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 analogous inequality for λ-terms is false: for example, the λ-term $\lambda \mathsf{x}^{\alpha}.\mathsf{x}^{\alpha}$ is of size $2 + |\alpha|$, while its sort $\alpha{\to}\alpha$ is of size $2 \cdot |\alpha| + 1$.

written in v. We start by the following lemma, saying that the flag counter cannot grow too much in a single place.

▶ **Lemma J.1.** *There exists a constant $C_\mathcal{G}$ such that for every $a \in A$, for every derivation tree t that derives $\varepsilon \vdash^\kappa_m \Lambda(\mathcal{G}) : \hat{\rho}^\kappa_m \triangleright d$ (for some d), and every node u of t, if $c_u(a) \geq C_\mathcal{G}$, then there exists a child v of u such that $c_v(a) \geq \frac{1}{C_\mathcal{G}} \cdot c_u(a)$.*

Proof. As $C_\mathcal{G}$ we take a number such that:

- $C_\mathcal{G} \geq (m+3)\cdot(r+1)$ for every subterm of $\Lambda(\mathcal{G})$ that is of the form $b\langle P_1, \dots, P_r\rangle$, and
- $C_\mathcal{G} > (m+3)\cdot(1+|A|\cdot(m+1))$.

Such $C_\mathcal{G}$ exists because $\Lambda(\mathcal{G})$ has finitely many subterms.

Fix now some $a \in A$, and consider a type judgment $\Gamma \vdash^\kappa_m R : (F, M, \tau) \triangleright c$ appearing in t, such that $c(a) \geq C_\mathcal{G}$. 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 $c(a) \geq C_\mathcal{G} > 0$. 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 $c(a) \geq \frac{1}{C_\mathcal{G}} \cdot c(a)$.

Suppose now that the considered type judgment is derived using one of the rules (Con0), (Con1), (Con≥1), i.e., that R is of the form $b\langle P_1, \dots, P_r\rangle$, where $b \neq \mathsf{nd}$. Let $\Gamma_i \vdash^\kappa_m P_i : (F_i, M_i, \tau_i) \triangleright c_i$ for $i \in \{1, \dots, r\}$ be the premisses of this rule. It holds that $(F, c) \in \mathit{Comp}_m(M; (\{(0, b)\}, \mathbf{0}), (F_1, c_1), \dots, (F_r, c_r))$. Consider the numbers $f_{k,a}$ and $f'_{k,a}$ as in the definition of the Comp_m predicate. We have that $f_{k,a} = f'_{k,a} + |\{(0, b)\} \cap \{(k, a)\}| + \sum_{i=1}^r |F_i \cap \{(k, a)\}| \leq f'_{k,a} + 1 + r$ for $k \in \{0, \dots, m+1\}$, and $f'_{k,a} \leq f_{k-1,a}$ for $k \in \{1, \dots, m+1\}$, and $f'_{0,a} = 0$. It follows that $f_{m+1,a} \leq (m+2)\cdot(r+1) \leq (m+2)\cdot(r+1)\cdot\frac{1}{C_\mathcal{G}}\cdot c(a)$ (the latter inequality holds because $c(a) \geq C_\mathcal{G}$). Suppose now, contrary to the thesis, that $c_i(a) < \frac{1}{C_\mathcal{G}} \cdot c(a)$ for all $i \in \{1, \dots, r\}$. Then we have that

$$c(a) = f_{m+1,a} + c_1(a) + \dots + c_r(a) \leq (m+2)\cdot(r+1)\cdot\frac{1}{C_\mathcal{G}}\cdot c(a) + r\cdot\frac{1}{C_\mathcal{G}}\cdot c(a)\,,$$

which gives

$$C_\mathcal{G} \leq (m+3)\cdot(r+1) - 1\,.$$

This contradicts the assumption that $C_\mathcal{G} \geq (m+3)\cdot(r+1)$, thus necessarily $c_i(a) \geq \frac{1}{C_\mathcal{G}} \cdot c(a)$ for some $i \in \{1, \dots, r\}$.

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

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

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

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

Next, suppose that $\max(\mathit{ord}(Q)+1,l) \le k \le m+1$. 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 $|A|\cdot(m+1)$, since $\sum_{i\in I} M_i \le \sum_{i\in\{0\}\cup I} M_i = M \in \mathcal{M}_m^{\kappa}$. It follows that there are at most $|A|\cdot(m+1)$ indices $i\in I$ for which $M_i \neq \mathbf{0}$. On the other hand, from Lemma G.4 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 $(k,a) \notin F_i{\restriction_{>\mathit{ord}(Q)}}$ (since $k \ge l$, we have $M(j) > 0$ for all $j \in \{k, k+1, \dots, m\}$, so $(k,a) \in F_i{\restriction_{>\mathit{ord}(Q)}}$ implies wildness). So (3) follows also in this case:

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

Using (3) for all possible k, and the assumption that $c(a) \ge C_{\mathcal{G}}$, we obtain that

$$f_{m+1,a} \le (m-l+2)\cdot(1+|A|\cdot(m+1)) \le (m+2)\cdot(1+|A|\cdot(m+1))\cdot\frac{1}{C_{\mathcal{G}}}\cdot c(a)\,.$$

Suppose now, contrary to the thesis, that $c_i(a) < \frac{1}{C_{\mathcal{G}}}\cdot c(a)$ for all $i \in \{0\}\cup I$. For every $i\in I$ such that $M_i = \mathbf{0}$, by Lemma G.3 (which can be applied because $\mathit{ord}(Q) \le m$) we actually have that $c_i(a) = 0$. There are at most $|A|\cdot(m+1)$ indices $i \in I$ for which $M_i \neq \mathbf{0}$, i.e., for which $c_i(a)$ can be nonzero (plus one more index $i=0$). We thus obtain:

$$\begin{aligned} c(a) &= f_{m+1,a} + \sum_{i\in\{0\}\cup I} c_i(a) \\ &\le (m+2)\cdot(1+|A|\cdot(m+1))\cdot\frac{1}{C_{\mathcal{G}}}\cdot c(a) + (1+|A|\cdot(m+1))\cdot\frac{1}{C_{\mathcal{G}}}\cdot c(a)\,, \end{aligned}$$

which gives

$$C_{\mathcal{G}} \le (m+3)\cdot(1+|A|\cdot(m+1))\,.$$

This contradicts the assumption that $C_{\mathcal{G}} > (m+3)\cdot(1+|A|\cdot(m+1))$, thus necessarily $c_i(a) \ge \frac{1}{C_{\mathcal{G}}}\cdot c(a)$ for some $i \in \{0\}\cup I$. ◀

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

▶ **Lemma J.2.** *There exists a pumpable derivation of $\varepsilon \vdash_m^{\kappa} \Lambda(\mathcal{G}) : \hat{\rho}_m^{\kappa} \triangleright c$ if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m^{\kappa} \Lambda(\mathcal{G}) : \hat{\rho}_m^{\kappa} \triangleright c_n$ with some c_n such that $c_n(a) \ge n$ for all $a \in A$.*

Proof. The left-to-right implication was already justified on page 11. We now prove the right-to-left implication. Let $C_{\mathcal{G}}$ be the constant from Lemma J.1, and let $K = |\mathcal{U}^{\mathcal{G}}_{/\sim}|$. Take $n = (C_{\mathcal{G}})^K$, and consider a derivation tree t that derives $\varepsilon \vdash_m^{\kappa} \Lambda(\mathcal{G}) : \hat{\rho}_m^{\kappa} \triangleright c_n$ for some c_n such that $c_n(a) \ge n$ for all $a \in A$; it exists by assumption. We claim that t is pumpable. In order to prove this, take some symbol $a \in A$. 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; in other words, whenever w is a child of u in t, it holds that $c_w(a) \le c_u(a)$. We will construct a sequence $v_0,\dots,v_K$ of K nodes lying on one branch in t, such that $c_{v_i}(a) \ge (C_{\mathcal{G}})^{K-i}$ for $i \in \{0,\dots,K\}$, and $c_{v_i}(a) < c_{v_{i-1}}(a)$ for $i \in \{1,\dots,K\}$. As v_0 we take the root of t; then $c_{v_0}(a) = c_n(a) \ge (C_{\mathcal{G}})^K$. Now suppose that $v_0,\dots,v_{i-1}$ are already constructed, and we want to construct v_i (we do this by induction, for $i = 1,2,\dots,K$). 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

$c_{v_i}(a) \geq \frac{1}{C_\mathcal{G}} \cdot c_u(a)$, which exists by Lemma J.1. Because $c_u(a) = c_{v_{i-1}}(a) \geq (C_\mathcal{G})^{K-i+1}$, it follows that $c_{v_i}(a) \geq (C_\mathcal{G})^{K-i}$; we also have $c_{v_i}(a) < c_{v_{i-1}}(a)$.

Once $v_0, \dots, v_K$ 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 $v_0, \dots, v_K$ 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 pumpable. ◀

J.3 Algorithms

We now give two algorithms which check whether a pumpable derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ exists for some c, and which in effect solves the diagonal problem. The 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 $|\mathcal{G}| + |A|$, the component $f(|A|)$ is anyway dominated by $|\mathcal{U}^\mathcal{G}_{/\sim}|$). The second algorithm is nondeterministic, and works in time polynomial in $|\mathcal{U}^\mathcal{G}_{/\sim}| + |\mathcal{G}| + |A|$, so it avoids the exponential dependence on $|A|$. The existence of these algorithms proves upper bounds required by Theorems 2 and 5.

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 type judgment appearing in J.

Our algorithm computes several sets, which we now define. 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$ 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 premisses). 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$ one can derive a type judgment $J \in \mathcal{U}^\mathcal{G}$, and $(J_l^0, K^0) \in \mathcal{E}$ for some $l \in \{1, \dots, r\}$, 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and $J_k^0 \in \mathcal{D}_a$ for some $k \in \{1, \dots, r\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$ 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$ one can derive a type judgment $J \in \mathcal{U}^\mathcal{G}$, and $(J_l^0, K^0) \in \mathcal{E}_a$ for some $l \in \{1, \dots, r\}$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$,
- if $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and $J_k^0 \in \mathcal{D}_a$ for some $k \in \{1, \dots, r\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ one can derive a type judgment $J \in \mathcal{U}^\mathcal{G}$, and $(J_l^0, K^0) \in \mathcal{E}$ for some $l \in \{1, \dots, r\} \setminus \{k\}$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$, and
- if by applying a rule to type judgments $J_1^0, \dots, J_r^0 \in \mathcal{D}$ one can derive a type judgment $J \in \mathcal{U}^\mathcal{G}$ satisfying $c_J(a) > 0$, and $(J_l^0, K^0) \in \mathcal{E}$ for some $l \in \{1, \dots, r\}$, then $(J{\downarrow}, K^0) \in \mathcal{E}_a$.

▶ **Lemma J.3.** *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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$, where from the induction assumption we know for $i \in \{1, \dots, r\}$ 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 $J_1, \dots, J_r$, and results in a type judgment J equivalent to J', where $c_J = c_{J'} + c_{J_1} + \dots + c_{J_r}$. If $J_k^0 \in \mathcal{D}_a$ for some $k \in \{1, \dots, r\}$ (first item in the definition of $\mathcal{D}_a$), by the induction assumption we actually know that $c_{J_k}(a) > 0$, so also $c_J(a) > 0$. If $c_{J'}(a) > 0$ (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 $J_1, \dots, J_r$ be its premisses. As already said, $J_1, \dots, J_r$ necessarily belong to $\mathcal{U}^{\mathcal{G}}$. By the induction assumption we have that $J_1{\downarrow}, \dots, J_r{\downarrow} \in \mathcal{D}$. The application of the final rule remains valid if we replace the flag counters in $J_1, \dots, J_r$ 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.

- If $c_{J_k}(a) > 0$ for some $k \in \{1, \dots, r\}$, then we have $J_k{\downarrow} \in \mathcal{D}_a$ 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 $i \in \{1, \dots, r\}$. Then, while replacing the flag counters in $J_1, \dots, J_r$ 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 actually very similar, and thus it is left to the reader. ◀

We now come to pumpable derivations. Here we need a few more definitions. For a set of symbols $B \subseteq A$, we say that a derivation is *B-pumpable* if for every symbol $a \in B$, there are two equivalent type judgments lying on one branch of the derivation and such that the a-coordinate of their flag counter differs (the previous notion of a pumpable derivation was for $B = A$). Next, for a nonempty subset B of A we define a B-skeleton (we will use skeletons to describe a general shape of a pumpable derivation). 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
- $(S_1), \dots, (S_s)$, where S_i is a B_i-skeleton for $i \in \{1, \dots, s\}$, for some division of B into disjoint nonempty subsets $B_1, \dots, B_s$, 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 $(\mathsf{b}), (\mathsf{a}), (\mathsf{c})$, and $(\mathsf{a}), ((\mathsf{b}), (\mathsf{c}))$. 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.

Below, we assume that premisses of a rule are always listed in some order, so that in the (Con≥1) rule consecutive premisses concern consecutive subterms $P_1, \dots, P_r$ of the considered

λ-term $a\langle P_1, \dots, P_r\rangle$, and in the (@) rule we first have a premiss concerning the function and then premisses concerning the argument (listed in any order).

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

- if $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and $J_k^0 \in \mathcal{P}_S$ for some $k \in \{1, \dots, r\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ 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 $(S_1), \dots, (S_s)$, and $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and there is a subsequence $J_{j_1}^0, \dots, J_{j_s}^0$ of $J_1^0, \dots, J_r^0$ (with $j_1 < \dots < j_s$) satisfying $J_{j_i}^0 \in \mathcal{P}_{S_i}$ for $i \in \{1, \dots, s\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ one can derive a type judgment $J \in \mathcal{U}^{\mathcal{G}}$, then $J{\downarrow} \in \mathcal{P}_S$.

▶ **Lemma J.4.** *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 for a type judgment equivalent to J^0. This is 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 $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and $J_k^0 \in \mathcal{P}_S$ for some $k \in \{1, \dots, r\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ 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 for a type judgment equivalent to J_k^0, and for $i \in \{1, \dots, r\} \setminus \{k\}$ by Lemma J.3(a) we have a derivation for a type judgment equivalent to J_i^0. We finish the derivation by applying the considered rule, and we obtain a B-pumpable derivation for 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 J.3(d) implies that there is a derivation for a type judgments J_1 equivalent to J^0 in which some J_2 equivalent to J^0 appears, where $c_{J_1}(a) > c_{J_2}(a)$. 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 J.3(d) implies that there is a derivation tree t for 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' of 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 $(S_1), \dots, (S_s)$, and $J_1^0, \dots, J_r^0 \in \mathcal{D}$, and there is a subsequence $J_{j_1}^0, \dots, J_{j_s}^0$ of $J_1^0, \dots, J_r^0$ (with $j_1 < \dots < j_s$) satisfying $J_{j_i}^0 \in \mathcal{P}_{S_i}$ for $i \in \{1, \dots, s\}$, and by applying a rule to $J_1^0, \dots, J_r^0$ one can derive a type judgment equivalent to J^0. Let $B = B_1 \cup \dots \cup B_s$, where S_i is a B_i-skeleton for $i \in \{1, \dots, s\}$. Then, for $i \in \{1, \dots, s\}$, by the induction assumption we have a B_i-pumpable derivation of a type judgment equivalent to $J_{j_i}^0$. Moreover, for $i \in \{1, \dots, r\} \setminus \{j_1, \dots, j_s\}$ by Lemma J.3(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 for 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. Let $J_1, \dots, J_r$ 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 $k \in \{1, \dots, r\}$ is B-pumpable. Then, by the induction assumption, $J_k{\downarrow} \in \mathcal{P}_S$ for some B-skeleton S. Moreover, $J_1{\downarrow}, \dots, J_r{\downarrow} \in \mathcal{D}$ by Lemma J.3(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 J.3(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 $J_{j_1}, \dots, J_{j_s}$ of $J_1, \dots, J_r$, and a division of B to disjoint nonempty subsets $B_1, \dots, B_s$ (where $s \geq 2$) such that for all $i \in \{1, \dots, s\}$ the subderivation resulting in J_{j_i} is B_i-pumpable. Then for all $i \in \{1, \dots, s\}$ by the induction assumption we have that $J_{j_i}{\downarrow} \in \mathcal{P}_{S_i}$ for some B_i-skeleton S_i. Moreover, $J_1{\downarrow}, \dots, J_r{\downarrow} \in \mathcal{D}$ by Lemma J.3(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 J.5.** *Given a set of basic type judgments $\mathcal{D} \subseteq \mathcal{U}^{\mathcal{G}}$, and its subsets $\mathcal{D}_1, \dots, \mathcal{D}_s \subseteq \mathcal{D}$, and a basic type judgment $J^0 \in \mathcal{U}^{\mathcal{G}}$, and (in the case of (b)) a symbol $a_+ \in A$, it can be decided in time polynomial in $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + |\mathcal{G}| + s$ whether there exist type judgments $J^0_1, \dots, J^0_r \in \mathcal{D}$ such that a subsequence $J^0_{j_1}, \dots, J^0_{j_s}$ of $J^0_1, \dots, J^0_r$ (with $j_1 < \dots < j_s$) satisfies $J^0_{j_i} \in \mathcal{D}_i$ for $i \in \{1, \dots, s\}$, and such that by applying a type system rule to $J^0_1, \dots, J^0_r$ (listed in this order) 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 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. In the proof we always assume that we have some $a_+ \in A$. Formally, this can be problematic when $A = \emptyset$, but it should be clear that also in this case we can obtain an algorithm for (a), by simply removing all fragments talking about a_+.

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 $J^0_1 \in \mathcal{D}$. For $s = 1$ we require that $J^0_1 \in \mathcal{D}_1$, 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 $c(a_+) > 0$.

In the (Con$\geq$1) and (@) rules it is useful to consider a predicate $\mathit{Comp}'_m(k, a, M, F')$ which is true if $k \in \{0, \dots, m+1\}$, and $a \in A$, and there exists $l \in \{0, \dots, k\}$ 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

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

contains exactly these sets F for which

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

Moreover, the set

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

contains exactly these sets F for which

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

Consider now the (Con≥1) rule, whose conclusion should be $\Gamma \vdash^{\kappa}_{m} b\langle P_1, \dots, P_r\rangle : (F, M, o) \triangleright c$ for some c. Its premisses should be of the form $\Gamma_i \vdash^{\kappa}_{m} P_i : (F_i, M_i, o) \triangleright \mathbf{0}$ for $i \in \{1, \dots, r\}$, where we have a big choice for Γ_i, M_i, F_i. The key point is that we do not need to know all premisses simultaneously. Indeed, after scanning through the first n premisses, the only things that we need to remember are:

- the union $\Gamma'_n = \Gamma_1 \sqcup \dots \sqcup \Gamma_n$,
- the sum $M'_n = M_1 + \dots + M_n$,
- the set F'_n of these $(k, a) \in F \cup \{(m+1, a_+)\}$ for which $\mathit{Comp}'_m(k, a, M, F_i)$ is satisfied for some $i \in \{1, \dots, n\}$, and
- the maximal number s_n such that a subsequence $J^0_{j_1}, \dots, J^0_{j_{s_n}}$ of the list of these n premisses satisfies $J^0_{j_i} \in \mathcal{D}_i$ for $i \in \{1, \dots, s_n\}$.

These tuples $(\Gamma'_n, M'_n, F'_n, s_n)$ satisfy $\Gamma'_n \leq \Gamma$, and $M'_n \leq M$, and $F'_n \subseteq F \cup \{(m+1, a_+)\}$, and $s_n \leq s$. The number of such tuples is at most $2 \cdot |\mathcal{U}^{\mathcal{G}}_{/\sim}|^3 \cdot (s+1)$, because all possible choices for Γ'_n (and similarly for M'_n and for $F'_n \cap F$) can appear in some type judgments from $\mathcal{U}^{\mathcal{G}}$. Thus in the algorithm we make a loop over $n \in \{1, \dots, r\}$, where after each step we remember the set of all tuples $(\Gamma'_n, M'_n, F'_n, s_n)$ that can be obtained after considering any choice of the first n premisses. For every such n we consider all possible candidates for the n-th premiss, and basing on the set of obtainable tuples $(\Gamma'_{n-1}, M'_{n-1}, F'_{n-1}, s_{n-1})$ we compute the set of obtainable tuples $(\Gamma'_n, M'_n, F'_n, s_n)$. Knowing which tuples $(\Gamma'_r, M'_r, F'_r, s_r)$ can be obtained after choosing all premisses, we can determine whether the considered conclusion can be derived in the required way. Such an algorithm is polynomial (we remark that r can be larger than $|\mathcal{U}^{\mathcal{G}}_{/\sim}|$, but surely $r \leq |\mathcal{G}|$).

For the (@) rule the situation is similar. Let $\Gamma \vdash^{\kappa}_{m} P\,Q : (F, M, \tau) \triangleright c$ be the considered conclusion (where c is not fixed). We need to have one premiss concerning P, so we can iterate over all candidates. Fix some such candidate $\Gamma_0 \vdash^{\kappa}_{m} P : (F_0, M_0, C {\to} \tau) \triangleright \mathbf{0}$. Having some number of premisses concerning Q, namely $\Gamma_i \vdash^{\kappa}_{m} Q : (F_i, M_i, \tau_i) \triangleright \mathbf{0}$ for $i \in \{1, \dots, n\}$, we only need to remember:

- 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 $\mathit{Comp}'_m(k, a, M, F_i{\restriction_{>\mathit{ord}(Q)}})$ is satisfied for some $i \in \{1, \dots, n\}$,
- the triple container $C' = \{\!|(F_i{\restriction_{\leq \mathit{ord}(Q)}}, M_i{\restriction_{\leq \mathit{ord}(Q)}}, \tau_i) \mid i \in \{1, \dots, n\}|\!\}$, and
- the maximal number s' such that a subsequence $J^0_{j_1}, \dots, J^0_{j_{s'}}$ of the list containing the premiss for P and the n premisses for Q satisfies $J^0_{j_i} \in \mathcal{D}_i$ for $i \in \{1, \dots, s'\}$.

We necessarily have that $\Gamma' \leq \Gamma$, and $M' \leq M$, and $F' \subseteq F \cup \{(m+1, a_+)\}$, and $C' \leq C$, and $s' \leq s$, so the number of possible tuples $(\Gamma', M', F', C', s')$ is at most $2 \cdot |\mathcal{U}^{\mathcal{G}}_{/\sim}|^4 \cdot s$. Having an obtainable tuple, and some candidate for a premiss, we can easily compute the tuple obtained after including this premiss. When this set is computed, we can easily determine whether the conclusion can be derived in the required way. ◀

Having established Lemma J.5, we come back to the main algorithm, where we want to compute all the sets. 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 $J^0_1, \dots, J^0_r$ 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 J.5(a) (where we take $s = 0$). Clearly, we will 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 $|\mathcal{U}^{\mathcal{G}}_{/\sim}| + |\mathcal{G}|$.

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 $J^0_1, \dots, J^0_r$ belonging to the current version of $\mathcal{D}$, where J^0_k for some $k \in \{1, \dots, r\}$ belongs to $\mathcal{D}_a$ (the first item of the definition); this can be done by Lemma J.5(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 $J^0_1, \dots, J^0_r$ belonging to the current version of $\mathcal{D}$ (the second item of the definition); this can be done by Lemma J.5(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 J.5(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 J.5, where $s \leq 2$.

Finally, we want to compute the set $\mathcal{P}_S$ for some skeleton S, assuming that we have already computed the set $\mathcal{P}_{S'}$ if $S = a[S']$, and the sets $\mathcal{P}_{S_1}, \dots, \mathcal{P}_{S_s}$ if S equals $(S_1), \dots, (S_s)$. 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 J.5. The fourth item is more complicated, but Lemma J.5(a) is perfectly suited to solve it.

At the end, due to the equivalence given by Lemma J.4, we need to check whether the type judgment $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright \mathbf{0}$ 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, compute $\mathcal{P}_S$ only for this skeleton (and recursively for its subskeletons), and check whether the type judgment belongs there. As already said, a single set $\mathcal{P}_S$ can be computed in polynomial time. 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.

J.4 Lower Bounds

We now prove lower bound appearing in Theorems 2 and 5. We base here on the problem of nonemptiness of $\mathcal{L}(\mathcal{G})$. The complexity of this problem was established by Engelfried [9] for higher-order pushdown automata, which are equivalent to a subclass of higher-order recursion schemes [14]. For us it is more convenient to use results of Kobayashi and Ong [19], as they talk directly about schemes.

Kobayashi and Ong [19, Theorem 4.3] prove that for $m \geq 1$ the following problem is m-EXPTIME-hard: given a scheme $\mathcal{G}$ of order at most $m+1$, and a disjunctive alternating parity tree automaton $\mathcal{B}$, decide whether $\mathcal{B}$ accepts $BT(\Lambda(\mathcal{G}))$. Instead of recalling the definition of a disjunctive alternating parity tree automaton, we notice that $\mathcal{G}$ and $\mathcal{B}$ produced by their reduction are of a special form. Namely, $BT(\Lambda(\mathcal{G}))$ consists of binary nodes labeled by br_2 and leaves labeled by e. The automaton $\mathcal{B}$ is always the same, and it accepts those trees, which contain an e-labeled leaf. Let us rename all br_2 symbols appearing in $\mathcal{G}$ to nd; call the resulting scheme $\mathcal{G}'$. Then $\mathcal{B}$ accepts $BT(\Lambda(\mathcal{G}))$ if and only if $\mathcal{L}(\mathcal{G}')$ is nonempty. Moreover, $\mathcal{G}'$ is word-recognizing. It follows that for $m \geq 1$ the following problem is m-EXPTIME-hard: given a word-recognizing scheme $\mathcal{G}'$ of order at most $m+1$, decide whether $\mathcal{L}(\mathcal{G}')$ is nonempty.

Kobayashi and Ong [19, Corollary 3.7] also 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 $BT(\Lambda(\mathcal{G}))$. In this reduction, 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 whose unique node is labeled by T,
- if L contains a tree T, then L contains every tree T' whose root is labeled by E, and such that the subtree rooted in one among children of the root equals T, and
- if L contains trees $T_1, \dots, T_n$, then L contains the tree T' whose root is labeled by A, and such that the subtrees rooted in children of the root are $T_1, \dots, T_n$.

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 symbols 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 for $m \geq 1$ the following problem is m-EXPTIME-hard: given a scheme $\mathcal{G}'$ of order at most m, decide whether $\mathcal{L}(\mathcal{G}')$ is nonempty.

For the special case of $A = \emptyset$, the diagonal problem checks precisely whether the language is nonempty, i.e., $\mathit{Diag}_\emptyset(\mathcal{L}(\mathcal{G}))$ holds if and only if $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Thus from the above we immediately obtain that for $m \geq 1$ the diagonal problem for word-recognizing order-$(m+1)$ schemes and the diagonal problem for tree-recognizing order-m schemes are m-EXPTIME-hard.

One may wonder whether the problems are m-EXPTIME-hard also when A is nonempty or, in particular, when A contains all letters appearing in the scheme. As expected, this is the case. Let us reduce from the problem of nonemptiness of $\mathcal{L}(\mathcal{G})$ to the problem of deciding $\mathit{Diag}_{\{\mathsf{a}\}}(\mathcal{L}(\mathcal{G}'))$, where $\mathcal{G}'$ is a scheme using only one symbol a (beside of the nd symbol). To produce $\mathcal{G}'$ we add to $\mathcal{G}$ a fresh starting nonterminal N_0' with 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 symbol 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 a symbols; thus $\mathit{Diag}_{\{\mathsf{a}\}}(\mathcal{L}(\mathcal{G}'))$ holds if and only if $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Moreover, the orders of $\mathcal{G}$ and $\mathcal{G}'$ are the same, and if $\mathcal{G}$ was word-recognizing, so is $\mathcal{G}'$.

In the last part, we prove that the diagonal problem for order-0 word-recognizing schemes is NP-hard. Of course this problem is a special case of the diagonal problem for order-1 word-recognizing schemes, and of the diagonal problem for order-0 tree-recognizing schemes, so the latter two problems are also NP-hard. We reduce from the undirected Hamiltonian cycle problem, known to be NP-complete [12]. We are thus given an undirected graph G, and we construct a word-recognizing scheme $\mathcal{G}$ of order 0 such that $\mathit{Diag}_A(\mathcal{G})$ holds if and only if there exists a Hamiltonian cycle in G. Suppose that the nodes of G are named $1, \dots, m$, and

$m \geq 2$. In $\mathcal{G}$ we use symbols $\mathsf{a}_1, \dots, \mathsf{a}_m$, and we take all of them to A. The nonterminals of $\mathcal{G}$ are N^i_j for $i \in \{0, \dots, m\}$ and $j \in \{1, \dots, m\}$, all of sort o. For $i, j \in \{1, \dots, m\}$ the rules are:

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

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

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

As the starting nonterminal we take N^m_1.

By induction on $i \in \{0, \dots, m\}$ we can see that $\mathcal{L}(BT(\Lambda_{\mathcal{G}}(\mathsf{N}^i_j)))$ contains words of the form $(\mathsf{a}_{j_i})^{n_i}(\mathsf{a}_{j_{i-1}})^{n_{i-1}} \dots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0}$, where $n_1, \dots, n_i$ are arbitrary positive numbers, and $j_i, j_{i-1}, \dots, j_0$ 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 $(\mathsf{a}_{j_m})^{n_m}(\mathsf{a}_{j_{m-1}})^{n_{m-1}} \dots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0}$, where $n_1, \dots, n_m$ are arbitrary positive numbers, and $j_m, j_{m-1}, \dots, j_0$ is a path in G such that $j_m = j_0 = 1$. If G contains Hamiltonian cycles, as $j_m, j_{m-1}, \dots, j_0$ we can take one of them, starting and ending in node 1 (by definition all Hamiltonian cycles have length m). Then the words $(\mathsf{a}_{j_m})^n(\mathsf{a}_{j_{m-1}})^n \dots (\mathsf{a}_{j_1})^n\mathsf{a}_{j_0}$ for all $n \geq 1$ are in $\mathcal{L}(\mathcal{G})$ and contain every symbol from $\mathsf{a}_1, \dots, \mathsf{a}_m$ at least n times, so they witness that $\mathit{Diag}_A(\mathcal{G})$ holds. Oppositely, if G does not contain Hamiltonian cycles, then on every path $j_m, j_{m-1}, \dots, j_0$ with $j_m = j_0 = 1$ some node $k \in \{2, \dots, m\}$ is missing. In consequence, in every word $(\mathsf{a}_{j_m})^{n_m}(\mathsf{a}_{j_{m-1}})^{n_{m-1}} \dots (\mathsf{a}_{j_1})^{n_1}\mathsf{a}_{j_0} \in \mathcal{L}(\mathcal{G})$ some letter $\mathsf{a}_k \in A$ does not appear at all, so $\mathit{Diag}_A(\mathcal{G})$ does not hold. This proves that the reduction is correct.

K Theorem 4

Finally, we prove Theorem 4. This theorem concerns the problem of deciding whether $I \subseteq \mathcal{L}(\mathcal{H})\!\downarrow$, where I is an ideal, and $\mathcal{H}$ is a word-recognizing scheme of order at most $m + 1$, where m is a fixed positive number.

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 polynomial time). Thus from Zetzsche [29] it follows that the problem of deciding whether $I \subseteq \mathcal{L}(\mathcal{H})\!\downarrow$ is NL-complete for $\mathcal{H}$ of order 0 (i.e., for $m = -1$), and P-complete for $\mathcal{H}$ of order at most 1 (i.e., for $m = 0$).

But here we assume that $m \geq 1$. Let us first see that the problem is m-EXPTIME-hard. This follows directly from m-EXPTIME-hardness of the problem of deciding whether $\mathcal{L}(\mathcal{H})$ is nonempty (cf. Appendix J.4). 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-EXPTIME. We follow here the approach of Zetzsche [28, 29]. He has shown [28, Proof of Theorem 1] that basing on an ideal I one can construct a nondeterministic finite-state transducer $\mathcal{T}$ and a set of symbols A such that for every language L we have that $I \subseteq L\!\downarrow$ if and only if $\mathit{Diag}_A(\mathcal{T}(L))$ holds.[11] 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

[11] 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 adopt the constructions proposed here to concrete formal definitions.

$v \in L$ 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 the diagonal problem for word-recognizing schemes of order at most $m + 1$ can be solved in m-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 symbol, 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 [27, 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 adopted. In particular, it works well when as $\mathcal{R}(N)$ we allow arbitrary λ-terms (cf. Appendix C.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 symbol as its label (and with at most one child), in $\mathcal{B}$ we allow to read this symbol, and if this symbol had no children, we accept. As a result of this construction we obtain an automaton $\mathcal{B}$ which 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 $BT(\Lambda(\mathcal{H}_{\mathcal{T}})) = T$ [11], i.e., $\mathcal{L}(\mathcal{H}_{\mathcal{T}}) = \mathcal{T}(\mathcal{L}(\mathcal{H}))$.
- 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 $m + 1$, as required.