The Diagonal Problem for Higher-Order Recursion Schemes is Decidable

Lorenzo Clemente * Paweł Parys †
University of Warsaw
Warsaw, Poland
{l.clemente,parys}@mimuw.edu.pl

Sylvain Salvati Igor Walukiewicz
University of Bordeaux, CNRS, INRIA
Bordeaux, France
{sylvain.salvati,igw}@labri.fr

Keywords downward closure, separability problem, diagonal problem, higher-order recursion schemes, higher-order OI grammars.

Abstract

A non-deterministic recursion scheme recognizes a language of finite trees. This very expressive model can simulate, among others, higher-order pushdown automata with collapse. We show decidability of the diagonal problem for schemes. This result has several interesting consequences. In particular, it gives an algorithm that computes the downward closure of languages of words recognized by schemes. In turn, this has immediate application to separability problems and reachability analysis of concurrent systems.

1. Introduction

The *diagonal problem* is a decision problem with a number of interesting algorithmic consequences. It is a central subproblem for computing the downward closure of languages of words [28], as well as for the problem of separability by piecewise-testable languages [11]. It is used in deciding reachability of a certain type of parameterized concurrent systems [26]. In its original formulation over finite words, the problem asks, for a given set of letters Σ and a given language of words L, whether for every number n there is a word in L where every letter from Σ occurs at least n times. In this paper, we study a generalization of the diagonal problem for languages of finite trees recognized by non-deterministic higher-order recursion schemes.

Higher-order recursion schemes are algorithmically manageable abstractions of higher-order programs. Higher-order features are now present in most mainstream languages like Java, JavaScript, Python, or C++. Higher-order schemes, or, equivalently, simply typed lambda-calculus with a fixpoint combinator, are a formalism that can faithfully model the control flow in higher-order programs. In this paper, we consider non-deterministic higher-order recursion schemes as recognizers of languages of finite trees. In other words we consider higher-order OI grammars [12, 22]. This is an expressive formalism covering many other models such as indexed grammars [2], ordered multi-pushdown automata [5], or the more general higher-order pushdown automata with collapse [17] (cf. also the equivalent model of ordered tree-pushdown automata [7]).

Our main result is a procedure for solving the diagonal problem for higher-order schemes. This is a missing ingredient to obtain several new decidability results for this model. It is well-known that schemes have a decidable emptiness problem [24], and it can be shown that they are closed under rational linear transductions, and in particular they form a full trio when restricted to finite word languages. In this context, a result by Zetzsche [28] entails computability of the *downward closure* of languages of words recognized by higher-order schemes. Moreover, a recent result by Czerwiński, Martens, van Rooijen, and Zeitoun [10] entails that the *separability by piecewise testable languages* is decidable for languages recognized by higher-order schemes. Finally, a third example comes from La Torre, Muscholl, and Walukiewicz [26] showing how to use downward closures to decide reachability in parameterized asynchronous shared-memory concurrent systems where every process is a higher-order scheme.

While the examples above show that the diagonal problem is intimately connected to downward closures[1], the computation of the downward closure is an important problem in its own right. The downward closure of a language offers an effective abstraction thereof. Since the subword relation is a well quasi-order [18], the downward closure of a language is always a regular language determined by a finite set of forbidden patterns. This abstraction is thus particularly interesting for complex languages, like those not having a semilinear Parikh image. While the downward closure is always regular, it is not always possible to effectively construct a finite automaton for it. This is obviously the case for classes with undecidable emptiness (since the downward closure preserves emptiness), but it is also the case for relatively better behaved classes for which the emptiness problem is decidable, such as Church-Rosser languages [13], and lossy channel systems [23].

The problem of computing the downward closure of a language has attracted a considerable attention recently. Early results show how to compute it for context-free languages [9, 27] (cf. also [4]), for Petri-net languages [14], for stacked counter automata [29], and context-free FIFO rewriting systems and $0L$-systems [1]. More recently, Zetzsche [28] has given an algorithm for indexed grammars, or equivalently for second-order pushdown automata. Hague, Kochems, and Ong [16] have made an important further advance by showing how to compute the downward closure of the language of pushdown automata of arbitrary order. In this

* This work was partially supported by the Polish National Science Centre grant 2013/09/B/ST6/01575.

† This work was partially supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

[1] In fact, the diagonal problem, separability by piecewise testable languages, and computing the downward closure are inter-reducible for full trios [11].

paper, we complete the picture by giving an algorithm for the more general model of higher-order pushdown automata with collapse [17]. We use the fact that these automata recognize the same class of languages as higher-order recursion schemes, and we work with the latter model instead.

Let us briefly outline our approach. While are mainly interested in higher-order recursion schemes (HORSes) generating finite words, for technical reasons we also need to consider narrow trees, i.e., trees with a bounded number of paths. In this we follow an idea of Hague et al. [16] who have used this technique for higher-order pushdown automata (without collapse). For a HORS $\mathcal{S}$ and a set of letters Σ, the diagonal problem asks whether for every $n \in \mathbb{N}$ there is a tree generated by $\mathcal{S}$ in which every letter from Σ appears at least n times. Our goal is an algorithm solving this problem. When $\mathcal{S}$ is of order 0, we have a regular grammar, for which the diagonal problem can be solved by direct inspection. For higher orders, apply a transformation that decreases the order by one. The order is decreased in two steps. First, we ensure that the HORS generates only narrow trees: we construct a HORS $\mathcal{S}'$, of the same order as $\mathcal{S}$, generating only narrow trees and such that the diagonal problems for $\mathcal{S}$ and $\mathcal{S}'$ are equivalent. Then, in the narrow HORS $\mathcal{S}'$ we lower the order by one: we create a HORS $\mathcal{S}''$ that is of order smaller by one than $\mathcal{S}'$ (but no longer narrow), and such that the diagonal problems for $\mathcal{S}'$ and $\mathcal{S}''$ are equivalent.

While narrowing the HORS is relatively easy to achieve, the main technical difficulty is order reduction. This point is probably better explained in terms of higher-order pushdown automata. If a higher-order pushdown automaton of order n accepts with an empty stack then an accepting computation has no choice but to pop out level-n stacks one by one. In other words, for every configuration the level-n return points are easily predictable. Using this we can eliminate them obtaining an automaton of order $n-1$. When we allow the collapse operation the situation changes completely: a configuration may have arbitrary many level-n return points, and different computations may use different return points.

In this paper we prefer to use HORSes rather than higher-order pushdown automata with collapse. Our solution resembles the one from [3], where a word-generating HORS is turned into a tree-generating HORS of order lower by one, whose frontier language (the language of words written from left to right in the leaves) is exactly the language of the original word-generating HORS. If our narrow trees were of width one (i.e., word-generating), we could just invoke [3], since their transformation preserves in particular the cardinality of the produced letters. While in general we need to handle narrow trees instead of words (a more general input than in [3]), we only prove that our construction preserves the number of their occurrences (and not their order, thus having a result weaker than in [3]). While the two results are thus formally incomparable, it is worth remarking that our construction does actually preserve the order of symbols belonging to the same branch of the narrow tree.

After some preliminaries in Section 2, we state formally our main result and some of its consequences in Section 3. The rest of the paper is devoted to the proof. In Section 4, we present a transformation of a scheme to a narrow one that preserves the order, and in Section 5 we present the reduction of a narrow scheme to a scheme of a smaller order (but not necessarily narrow). Both reductions preserve the diagonal problem. Finally, in Section 6, we conclude with some further considerations.

2. Preliminaries

Higher-order recursion schemes. We use the name "sort" instead of "simple type" or "type" to avoid confusion with the types introduced later. The set of *sorts* is constructed from a unique basic sort o using a binary operation $\to$. Thus o is a sort, and if α, β are sorts, so is $\alpha \to \beta$. The order of a sort is defined by: $ord(o) = 0$, and $ord(\alpha \to \beta) = \max(1 + ord(\alpha), ord(\beta))$. By convention, $\to$ associates to the right, i.e., $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$. Every sort α can be uniquely written as $\alpha_1 \to \alpha_2 \to \ldots \to \alpha_n \to o$. The sort $o \to \cdots \to o \to \alpha$ with r occurrences of o is denoted $o^r \to \alpha$, where $o^0 \to \alpha$ is simply α.

The set of *terms* is defined inductively as follows. For each sort α there is a countable set of *variables* $x^\alpha, y^\alpha, \ldots$ and a countable set of *nonterminals* $A^\alpha, B^\alpha, \ldots$; all of them are terms of sort α. There is also a countable set of *letters* $a, b, \ldots$; out of a letter a and a sort α of order at most 1 one can create a *symbol* a^α that is a term of sort α. Moreover, if K and L are terms of sort $\alpha \to \beta$ and α, respectively, then $(K\,L)^\beta$ is a term of sort β. For $\alpha = (o^r \to o)$ we often shorten a^α to a^r, and we call r the *rank* of a^r. Moreover, we omit the sort annotation of variables, nonterminals, or terms, but note that each of them is implicitly assigned a particular sort. We also omit some parentheses when writing terms and denote $(\ldots(K\,L_1)\ldots L_n)$ simply by $K\,L_1 \ldots L_n$. A term is called *closed* if it uses no variables.

We deviate here from usual definitions in the detail that letters itself are unranked, and thus out of a single letter a one may create a symbol a^r for every rank r. This is convenient for us, as during the transformations of HORSes described in Sections 4 and 5 we need to change the rank of tree nodes, without changing their labels. Notice, however, that in terms a letter is used always with a particular rank.

A *higher-order recursion scheme* (HORS for short) is a pair $\mathcal{S} = (A_{init}, \mathcal{R})$, where A_{init} is the *initial nonterminal* that is of sort o, and $\mathcal{R}$ is a finite set of rules of the form $A^\alpha\, x_1^{\alpha_1}\, \ldots\, x_k^{\alpha_k} \to K^o$ where $\alpha = \alpha_1 \to \cdots \to \alpha_k \to o$ and K is a term that uses only variables from the set $\{x_1^{\alpha_1}, \ldots, x_k^{\alpha_k}\}$. The order of $\mathcal{S}$ is defined as the highest order of a nonterminal for which there is a rule in $\mathcal{S}$. We write $\mathcal{R}(\mathcal{S})$ to denote the set of rules of a HORS $\mathcal{S}$. Observe that our schemes are *non-deterministic* in the sense that $\mathcal{R}(\mathcal{S})$ can have many rules with the same nonterminal on the left side. A scheme with at most one rule for each nonterminal is called *deterministic*.

Let us now describe the dynamics of HORSes. Substitution is defined as expected:

$$A[M/x] = A, \qquad a^r[M/x] = a^r, \qquad x[M/x] = M,$$

$$y[M/x] = y \text{ if } y \neq x, \qquad (K\,L)[M/x] = K[M/x]\,L[M/x].$$

We shall use the substitution only when M is closed, so there is no need to perform α-conversion. We also allow simultaneous substitutions: we write $K[M_1/x_1, \ldots, M_k/x_k]$ to denote the simultaneous substitution of $M_1, \ldots, M_k$ respectively for $x_1, \ldots, x_k$. We notice that when the terms M_i are closed, this amounts to apply the substitutions $[M_i/x_i]$ (with $i \in \{1, \ldots, k\}$) in any order.

A HORS $\mathcal{S}$ defines a reduction relation $\to_{\mathcal{S}}$ on closed terms:

$$\frac{(A\,x_1\ \ldots\ x_k \to K) \in \mathcal{R}(\mathcal{S})}{A\,M_1\ \ldots\ M_k \to_{\mathcal{S}} K[M_1/x_1, \ldots, M_k/x_k]}$$

$$\frac{K_l \to_{\mathcal{S}} K_l' \text{ for some } l \in \{1, \ldots, r\} \qquad K_i = K_i' \text{ for all } i \neq l}{a^r\,K_1\ \ldots\ K_r \to_{\mathcal{S}} a^r\,K_1'\ \ldots\ K_r'}$$

We thus apply some of the rules of $\mathcal{S}$ to one of the outermost nonterminals in the term.

We are interested in finite trees generated by HORSes. A closed term L of sort o is a *tree* if it does not contain any nonterminal. A HORS $\mathcal{S}$ *generates* a tree L from a term K if $K \to_{\mathcal{S}}^* L$; when we do not mention the term K we mean generating from the initial nonterminal of $\mathcal{S}$. Since a scheme may have more than one rule for some nonterminals, it may generate more than one tree. We can view a HORS of order 0 essentially as a finite tree automaton, thus a HORS of order 0 generates a regular language of finite trees.

Let Δ be a finite set of symbols of rank 0 (called also *nullary* symbols). A tree K is Δ-*narrow* if it has exactly $|\Delta|$ leaves, each

of them labeled by a different symbol from Δ. A HORS is called Δ-*narrow* if it generates only Δ-narrow trees, and it is called *narrow* if it is Δ-narrow for some Δ. We are particularly interested in Δ-narrow HORSes for $|\Delta| = 1$; trees generated by them consist of a single branch and thus can be seen as words.

Transductions. A (bottom-up, nondeterministic) *finite tree transducer* (FTT) is a tuple $\mathcal{A} = (Q, Q_F, \delta)$, where Q is a finite set of control states, $Q_F \subseteq Q$ is the set of final states, and δ is a finite set of transitions of the form

$$a^r\,(p_1, x_1)\,\ldots\,(p_r, x_r) \longrightarrow q, t \quad \text{or}$$
$$p, x_1 \longrightarrow q, t \qquad (\varepsilon\text{-transition})$$

where a is a letter, $p, q, p_1, \ldots, p_r$ are states, $x_1, \ldots, x_r$ are variables of sort o, and t is a term built of variables from $\{x_1, \ldots, x_k\}$ ($\{x_1\}$, respectively) and symbols, but no nonterminals. An FTT $\mathcal{A}$ defines in a natural way a binary relation $\mathcal{T}(\mathcal{A})$ on trees [8]. We say that an FTT is *linear* if no term t on the right of transitions contains more than one occurrence of the same variable.

We show that HORSes are closed under linear transductions. The construction relies on the reflection operation [6], in order to detect unproductive subtrees.

Theorem 2.1. *HORSes are effectively closed under linear tree transductions.*

A family of word languages is a *full trio* if it is effectively closed under rational (word) transductions. Since rational transductions on words are a special case of linear tree transductions, we obtain the following corollary of Theorem 2.1.

Corollary 2.2. *Languages of finite words recognized by HORSes form a full trio.*

3. The Main Result

We formulate the main result and state some of its consequences.

Definition 3.1 (Diagonal problem)**.** *For a higher-order recursion scheme $\mathcal{S}$, and a set of letters Σ, the predicate $Diag_\Sigma(\mathcal{S})$ holds if for every $n \in \mathbb{N}$ there is a tree t generated by S with at least n occurrences of every letter from Σ. The* diagonal problem *for schemes is to decide whether $Diag_\Sigma(\mathcal{S})$ holds for a given scheme $\mathcal{S}$ and a set Σ.*

Theorem 3.1. *The diagonal problem for higher-order recursion schemes is decidable.*

Proof. The proof is by induction on the order of a HORS $\mathcal{S}$. It relies on results from the next two sections. If $\mathcal{S}$ has order 0, then $\mathcal{S}$ can be converted to an equivalent finite automaton on trees, for which the diagonal problem can be solved by direct inspection. For $\mathcal{S}$ of order greater than 0, we first convert $\mathcal{S}$ to a narrow HORS $\mathcal{S}'$ such that $Diag_\Sigma(\mathcal{S})$ holds iff $Diag_\Sigma(\mathcal{S}')$ holds (Theorem 4.1). Then, we employ the construction from Section 5 and obtain a HORS $\mathcal{S}''$ of order smaller by 1 than the order of $\mathcal{S}'$. By Lemmata 5.1 and 5.2: $Diag_\Sigma(\mathcal{S}')$ holds iff $Diag_\Sigma(\mathcal{S}'')$ holds. □

The main theorem allows to solve some other problems for higher-order schemes. The *downward closure* of a language of words is the set of its (scattered) subwords. Since the subword relation is a well quasi-order [18], the downward closure of any language of words is regular. The main theorem implies that the downward closure can be computed for HORSes generating languages of finite words, or, in our terminology, $\{e^0\}$-narrow HORSes, where e^0 is a nullary symbol acting as an end-marker.

Corollary 3.2. *There is an algorithm that given an $\{e^0\}$-narrow HORS $\mathcal{S}$ computes a regular expression for the downward closure of the language generated by $\mathcal{S}$.*

Proof. By Corollary 2.2, word languages generated by schemes are closed under rational transductions. In this case, Theorem 3.1 together with a result of Zetzsche [28] can be used to compute the downward closure of a language generated by a HORS. □

Piecewise testable languages of words are boolean combinations of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_k \Sigma^*$ for some $a_1, \ldots, a_k \in \Sigma$. Such languages talk about possible orders of occurrences of letters. The problem of separability by piecewise testable languages asks, for two given languages of words, whether there is a piecewise testable language of words containing one language and disjoint from the other. A separating language provides a simple explanation of the disjointness of the two languages [19].

Corollary 3.3. *There is an algorithm that given two $\{e^0\}$-narrow HORSes decides whether there is a piecewise testable language separating the languages of the two HORSes.*

Proof. This is an immediate consequence of a result of Czerwiński et al. [11] who show that for any class of languages effectively closed under rational transductions, the problem reduces to solving the diagonal problem. □

The final example concerns deciding reachability in parameterized asynchronous shared-memory systems [15]. In this model one instance of a process, called leader, communicates with an undetermined number of instances of another process, called contributor. The communication is implemented by common registers on which the processes can perform read and write operations; however, operations of the kind of test-and-set are not possible. The reachability problem asks if for some number of instances of the contributor the system has a run writing a designated value to a register.

Corollary 3.4. *The reachability problem for parameterized asynchronous shared-memory systems is decidable for systems where leaders and contributors are given by $\{e^0\}$-narrow HORSes.*

Proof. La Torre et al. [26] show how to use the downward closure of the language of the leader to reduce the reachability problem for a parameterized system to the reachability problem for the contributor. Being a full trio is sufficient for this reduction to work. □

4. Narrowing the HORS

The first step in our proof of Theorem 3.1 is to convert a scheme to a narrow scheme. The property of being narrow is essential for the second step, as lowering the order of a scheme works only for narrow schemes. This approach through narrowing has been used by Hague et al. [16] for higher-order pushdown automata. Here we deal with recursion schemes, which are equivalent to higher-order pushdown automata with collapse.

The idea behind narrowing is quite intuitive. Consider a binary tree, and suppose that we are interested in the number of occurrences of a certain letter a, that may appear only in leaves. Consider a path that, at each node, selects the subtree containing the larger number of a's, and let's label the node by a if the successor of the node that is not on the path has an a-labeled descendant. Then, if the original tree had n occurrences of a, then on the selected path we put between $\log n$ and n labels a. The lower bound holds since, whenever a subtree is selected, at most half of the a's is discarded (on the other subtree), and this happens a number of times equal to the number of a's on the resulting path. This observation implies it suffices to convert a scheme $\mathcal{S}$ generating trees to a scheme $\mathcal{S}'$ generating all paths (words) in the trees generated by $\mathcal{S}$ with the additional labeling. Then $Diag_{\{a\}}(\mathcal{S})$ will be equivalent to $Diag_{\{a\}}(\mathcal{S}')$.

The general situation is a bit more complicated since we are interested in the diagonal problem not just for a single letter, but for

a set of letters Σ. In this case, different letters may have different witnessing paths, so $\mathcal{S}'$ should generate not a single path but a narrow tree whose number of paths is bounded by $|\Sigma|$.

Theorem 4.1. *For a HORS $\mathcal{S}$ and a set of letters Σ, one can construct a set of nullary symbols Δ of size $|\Sigma|$ and a Δ-narrow HORS $\mathcal{S}'$ of the same order as $\mathcal{S}$, such that $Diag_\Sigma(\mathcal{S})$ holds if, and only if, $Diag_\Sigma(\mathcal{S}')$ holds.*

Proof. We start by assuming that $\mathcal{S}$ uses only symbols of rank 2 and 0, where additionally letters from Σ appear only in leaves. The general situation can be easily reduced to this one, by applying a tree transduction that replaces every node by a small fragment of a tree built of binary symbols, with the original label in a leaf.

Then, we consider a linear bottom-up transducer $\mathcal{A}$ from trees produced by $\mathcal{S}$ to narrow trees. As labels in the resulting trees we use: (i) new leaf symbols $\Delta = \{e_1^0, \dots, e_{|\Sigma|}^0\}$, (ii) unary symbols a^1 for all $a \in \Sigma$, and (iii) new auxiliary symbols $\bullet^k$ (of rank $k \geq 1$). For each set of letters $\Gamma \subseteq \Sigma$, $\mathcal{A}$ contains a state $p^?_\Gamma$ making sure that each letter from Γ occurs at least once in the input tree. Moreover, for each nonempty set of leaf labels $\Delta' \subseteq \Delta$, $\mathcal{A}$ contains a state $p_{\Delta'}$ that outputs only Δ'-narrow trees. The final state of $\mathcal{A}$ is p_Δ. Transitions are as follows:

$$\begin{aligned}
&\text{(Branch)} && a^2\,(p_{\Delta_1}, x_1)\,(p_{\Delta_2}, x_2) \longrightarrow p_{\Delta_1\cup\Delta_2}, \bullet^2\, x_1\, x_2\,,\\
&\text{(Leaf)} && a^0 \longrightarrow p_{\{e_{i_1},\dots,e_{i_k}\}}, \bullet^k\, e_{i_1}\ \dots\ e_{i_k}\,,\\
&\text{(Choose}_1\text{)} && a^2\,(p_{\Delta_1}, x_1)\,(p^?_\Gamma, x_2) \longrightarrow p_{\Delta_1}, a^1_1(\cdots(a^1_k\, x_1))\,,\\
&\text{(Choose}_2\text{)} && a^2\,(p^?_\Gamma, x_1)\,(p_{\Delta_2}, x_2) \longrightarrow p_{\Delta_2}, a^1_1(\cdots(a^1_k\, x_2))\,.
\end{aligned}$$

where Δ_1 and Δ_2 are *disjoint* subsets of Δ, where $i_1 < \cdots < i_k$, and where $\Gamma = \{a_1, \dots, a_k\} \subseteq \Sigma$. Intuitively, rules of types (Branch) and (Leaf) make sure that we output narrow trees, and rules of types (Choose$_i$) select a branch and output (only) letters that appear at least once in the discarded subtree. States $p^?_\Gamma$ check that each letter in Γ occurs at least once, as follows:

$$\begin{aligned}
&\text{(Check}_2\text{)} && a^2\,(p^?_{\Gamma_1}, x_1)\,(p^?_{\Gamma_2}, x_2) \longrightarrow p^?_{\Gamma_1\cup\Gamma_2}, e_1^0\\
&\text{(Check}_0\text{)} && a^0 \longrightarrow p^?_{\{a\}}, e_1^0
\end{aligned}$$

The set $\mathcal{T}(p^?_\Gamma)(\{t\})$ is either a single leaf or $\emptyset$, depending on whether t satisfies the condition or not. The choice of e_1^0 on the right side of the transitions is not important, since, in the way states $p^?_\Gamma$ are used, it only matters whether the input can be successfully parsed, and not what the output actually is.

It is clear that the image of state $p_{\Delta'}$ is always a language of Δ'-narrow trees. Correctness follows from the following claim.

Claim. *Let t be an input tree. Then, (i) if t has at least n occurrences of every letter $a \in \Sigma$, then $\mathcal{T}(\mathcal{A})(t)$ contains a tree with at least $\log n$ occurrences of every letter $a \in \Sigma$, and (ii) if $\mathcal{T}(\mathcal{A})(t)$ contains a tree with at least n occurrences of every letter $a \in \Sigma$, then t has at least n occurrences of every letter $a \in \Sigma$.*

To conclude the proof, let T be the transduction $\mathcal{T}(\mathcal{A})$ realized by $\mathcal{A}$. By Theorem 2.1, there exists a HORS $\mathcal{S}'$ of the same order as $\mathcal{S}$ with $\mathcal{L}(\mathcal{S}') = T(\mathcal{L}(\mathcal{S}))$. First, it is clear that $\mathcal{L}(\mathcal{S}')$ is a language of Δ-narrow trees. Second, thanks to the claim above, $Diag_\Sigma(\mathcal{S})$ holds if, and only if, $Diag_\Sigma(\mathcal{S}')$ holds. □

5. Lowering the Order

Let $\mathcal{S}$ be a Δ-narrow HORS of order $k \geq 1$, and let Σ be a finite set of letters. The goal of this section is to construct a HORS $\mathcal{S}'$ of order $k-1$ s.t. $Diag_\Sigma(\mathcal{S})$ holds if and only if $Diag_\Sigma(\mathcal{S}')$ holds.

Let $\bullet$ be a fresh letter, not used in $\mathcal{S}$, and not in Σ. We will use it to label auxiliary nodes of trees generated by $\mathcal{S}'$. We say that two trees K_1, K_2 are *equivalent* if, for each letter $a \neq \bullet$, they have the same number of occurrences of a. The resulting HORS $\mathcal{S}'$ will have the property that for every tree generated by $\mathcal{S}$ there exists an equivalent tree generated by $\mathcal{S}'$, and for every tree generated by $\mathcal{S}'$ there exists an equivalent tree generated by $\mathcal{S}$. Then surely $Diag_\Sigma(\mathcal{S})$ holds if and only if $Diag_\Sigma(\mathcal{S}')$ holds.

Let us explain the idea of lowering the order of a scheme on two simple examples. Consider the following transformation on sorts that removes arguments of sort o:

$$o{\downarrow} = o, \quad \text{and} \quad (\beta \to \gamma){\downarrow} = \begin{cases} \gamma{\downarrow} & \text{if } \beta = o,\\ (\beta{\downarrow}) \to (\gamma{\downarrow}) & \text{otherwise.}\end{cases}$$

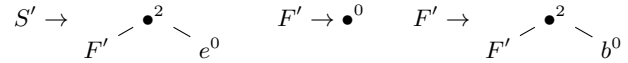
We have that the order of $\alpha{\downarrow}$ is $\max(0, ord(\alpha) - 1)$.

Very roughly our construction will take a scheme and produce a scheme of a lower order by changing every nonterminal of sort α to a nonterminal of sort $\alpha{\downarrow}$. This is achieved by outputting immediately arguments of sort o instead of passing them to nonterminals.

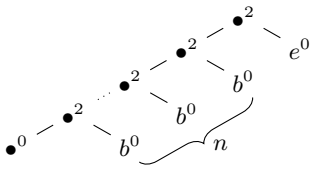
Example 1. Consider the scheme

$$S \to F\, e^0, \qquad F\, x \to x, \qquad F\, x \to F\,(b^1\, x)\,.$$

This scheme generates words of the form $(b^1)^n e^0$. It can be transformed to an equivalent scheme:



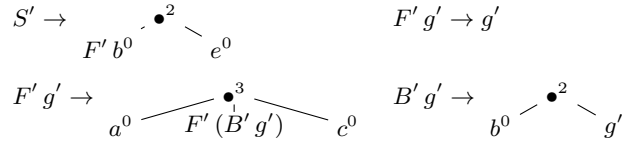
where we have used a graphical notation for terms; in standard notation the first rule would be $S' \to \bullet^2\, F'\, e^0$. Now both b and e are used with rank 0; we have also used auxiliary symbols $\bullet^2$ and $\bullet^0$. Observe that the new scheme has smaller order as the sorts of S' and F' are o. The new scheme is equivalent to the initial one since a derivation of $(b^1)^n e^0$ can be matched by the derivation of a tree with one e^0 and b^0 appearing n times:



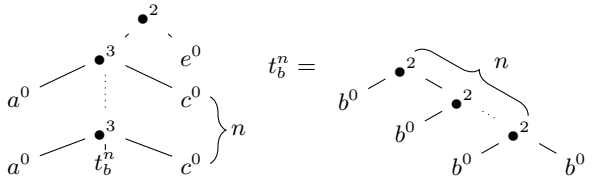
Example 2. Let us now look at a more complicated example. This time we take the following scheme of order 2:

$$S \to F\, b^1\, e^0, \quad F\, g\, x \to g\, x, \quad F\, g\, x \to a^1\,(F\,(B\, g)\,(c^1\, x)),$$
$$B\, g\, x \to b^1\,(g\, x)\,.$$

Here g has sort $o \to o$, and x has sort o. This scheme generates words of the form $(a^1)^n (b^1)^{n+1} (c^1)^n e^0$. We transform it into a scheme of order 1:



The latter scheme generates trees of the form:



The intuition behind the above two examples is as follows. Consider some closed term K of sort o, and its subterm L of sort o. In a tree generated by K, the term L will be used to generate some subtrees. Take a tree where L generates exactly k subtrees. Then we can create a new term starting with a symbol $\bullet^{k+1}$: in the first subtree we put K with L replaced by $\bullet^0$, and in the k remaining subtrees we put L. From this new term we can generate a tree similar to the initial one: the subtrees generated by L are moved closer to the root, but the multisets of letters appearing in the tree do not change. We do this with every subterm of sort o on the right hand side of every rule of $\mathcal{S}$. In the obtained system, whenever an argument has sort o then it is $\bullet^0$. Because of this, we can just drop arguments of sort o. This is what our translation $\alpha\downarrow$ on sorts does, and this is what happens in the two examples above. Since the original schemes from the two examples generated words, and all arguments were eventually used to generate a subword, for every subterm of sort o the multiplication factor k was always 1.

The crucial part of this argument was the information on the number of times L will be used in K. This is the main technical problem we need to address. We propose a special type system for tracking the use of closures of sort o. It will non-deterministically guess the number of usages, and then enforce derivations that conform to this guess. The reason why such a *finite* type system can exist is that $\mathcal{S}$ is Σ_0-narrow, which, in turn, implies that L can be used to generate at most $|\Sigma_0|$ subtrees of a tree.

In the sequel we assume w.l.o.g. that in $\mathcal{S}$ the only rule from the initial nonterminal is $A_{init} \to A\, e_1^0 \dots e_{|\Delta|}^0$ (for some nonterminal A) where $\Delta = \{e_1^0, \dots, e_{|\Delta|}^0\}$, and no other rule uses a nullary symbol nor the initial nonterminal A_{init}. To ensure this condition, we perform the following simple transformation of the HORS. Every rule $B\, x_1 \dots x_k \to K$ in $\mathcal{R}(\mathcal{S})$ is replaced by $B\, y_1 \dots y_{|\Delta|}\, x_1 \dots x_k \to K'$, where K' is obtained by replacing in K every use of a symbol $e_i^0 \in \Delta$ by y_i, and every use of a nullary symbol not being in Δ by an arbitrary y_i (this symbol anyway does not appear in any tree generated by $\mathcal{S}$), and every use of a nonterminal C by $C\, y_1 \dots y_{|\Delta|}$ (the sort of every nonterminal is changed from α to $o^{|\Delta|} \to \alpha$). Additionally a new rule $A_{init} \to A\, e_1^0 \dots e_{|\Delta|}^0$ is added, where A_{init} is a fresh nonterminal that becomes initial, and A is the nonterminal that was initial previously. It is easy to see that this transformation does not change the set of generated trees. It also does not increase the order, since in this section we assume that $\mathcal{S}$ has order at least 1.

5.1 Type System

We now present a type system whose main purpose is to track nullary symbols that eventually will end as leaves of a generated tree. The type of a term will say which nullary symbols are already present in the term and which will come from each of its arguments.

For every sort $\alpha = (\alpha_1 \to \dots \to \alpha_k \to o)$ we define the set $\mathcal{T}^\alpha$ of *types* of sort α and the set $\mathcal{LT}^\alpha$ of *labeled types* of sort α by induction on α. Labeled types in $\mathcal{LT}^\alpha$ are just pairs $(S,\tau) \in \mathcal{P}(\Delta) \times \mathcal{T}^\alpha$, where if $\alpha = o$ we require that $S \neq \emptyset$. The support of a set Λ of labeled types is the subset $\Lambda^{\neq\emptyset}$ of its elements $(S,\tau) \in \Lambda$ with $S \neq \emptyset$. A set of labeled types Λ is *separated* if there are no two distinct (S,τ) and (S',τ') in Λ s.t. $S \cap S' \neq \emptyset$. Types in $\mathcal{T}^\alpha$ are of the form $\Lambda_1 \to \dots \to \Lambda_k \to \mathbf{r}$, where $\mathbf{r}$ is a distinguished type corresponding to sort o, Λ_i is a subset of $\mathcal{LT}^{\alpha_i}$ for each $i \in \{1,\dots,k\}$ s.t. $\{\Lambda_1^{\neq\emptyset}, \dots, \Lambda_k^{\neq\emptyset}\}$ are pairwise disjoint and $\Lambda_1 \cup \dots \cup \Lambda_k$ is separated. Let us emphasize that Λ_i for $\alpha_i = o$ can only contain pairs (S,τ) with $S \neq \emptyset$. We fix some (arbitrary) order $<$ on elements of $\mathcal{LT}^\alpha$ for every sort α.

Types do not describe all the possible trees generated by a term, but rather restrict the generating power of a term. Intuitively, a labeled type $(S_0, \mathbf{r})$ assigned to a closed term of sort o says that we are interested in generating trees that are S_0-narrow. A functional type $(S_0, \Lambda \to \tau)$ says that the term becomes of type (S,τ) when taking an argument that will be used only with labeled types from Λ. Here, S equals S_0 plus the symbols $S_1 \cup \dots \cup S_k$ generated by an argument of type $\Lambda = \{(S_1,\tau_1), \dots, (S_k,\tau_k)\}$.

A *type environment* Γ is a set of bindings of variables of the form $x^\alpha : \lambda$, where $\lambda \in \mathcal{LT}^\alpha$; we may have multiple bindings $x^\alpha : \lambda_1, \dots, x^\alpha : \lambda_n$ for the same variable (which we also abbreviate as $x^\alpha : \{\lambda_1, \dots, \lambda_n\}$), however $\{\lambda_1, \dots, \lambda_n\}$ must be separated in the sense above. A *type judgment* is of the form $\Gamma \vdash M^\alpha : \lambda$, where again $\lambda \in \mathcal{LT}^\alpha$.

The rules of the type system are given in Figure 1. A *derivation* is a tree whose nodes are labeled by type judgments constructed according to the rules of the type system (we draw a parent below its children, unlikely the usual convention for trees). For the proof it will be convenient to assume that a derivation is an ordered tree: in the application rule the premise with L is the first sibling followed by the premises with M ordered using our fixed ordering on (S_i,τ_i), without repetitions. We say that D is a *derivation for* $\Gamma \vdash M : \lambda$, or that D *derives* $\Gamma \vdash M : \lambda$, if this type judgment labels the root of D. All the nodes of derivations are required to be labeled by valid type judgments, thus all the restrictions on types from the definition of $\mathcal{T}^\alpha$ stay in force; in particular, in the application rule for LM, the sets $S_1, \dots, S_k$ are disjoint.

5.2 Transformation

Once we have the type system, we can show how the HORS $\mathcal{S}$ is transformed into the HORS $\mathcal{S}'$.

A term of type τ will be transformed into a term of sort $tr(\tau)$. This sort is defined by induction on the structure of τ, as follows:

- $tr(\mathbf{r}) = o$, and
- if $\tau = (\Lambda \to \tau') \in \mathcal{T}^{\alpha\to\beta}$ with $\Lambda = \{(S_1,\tau_1) < \dots < (S_k,\tau_k)\}$, then we have

$$tr(\tau) = \begin{cases} tr(\tau_1) \to \dots \to tr(\tau_k) \to tr(\tau') & \text{if } \alpha \neq o, \\ tr(\tau') & \text{if } \alpha = o. \end{cases}$$

We see that if $\tau \in \mathcal{T}^\alpha$, then $ord(tr(\tau)) = \max(0, ord(\alpha) - 1)$. This translation is a refined version of the translation $\alpha\downarrow$ on sorts that we have seen earlier in the examples.

The nonterminals of $\mathcal{S}'$ will be the nonterminals of $\mathcal{S}$ labeled with types. For every nonterminal A from $\mathcal{S}$, of some sort α, and for every τ such that $(\emptyset,\tau) \in \mathcal{LT}^\alpha$, in $\mathcal{S}'$ we consider a nonterminal $A{\uparrow_\tau}$ of sort $tr(\tau)$. Moreover, for every variable x used in $\mathcal{S}$, being of some sort $\alpha \neq o$, and for every $\lambda = (S,\tau) \in \mathcal{LT}^\alpha$, in $\mathcal{S}'$ we consider a variable $x{\uparrow_\lambda}$ of sort $tr(\tau)$.

Before defining the rules of $\mathcal{S}'$, we need to explain how to transform terms to match the transformation on types. This transformation is guided by derivations. We define a term $tr(D)$, where D is a derivation for $\Gamma \vdash K : \lambda$, as follows:

- If $K = a^r$ is a symbol, then $tr(D) = a^0$.
- If $K = x^\alpha$ is a variable, then $tr(D) = \bullet^0$ if $\alpha = o$, and $tr(D) = x{\uparrow_\lambda}$ otherwise.
- If $K = A$ is a nonterminal, then $tr(D) = A{\uparrow_\tau}$ provided that $\lambda = (\emptyset,\tau)$.
- Suppose that $K = L\, M$ is an application. Then in D we have a subtree D_0 deriving $\Gamma \vdash L : (S_0, \Lambda \to \tau)$, where $\Lambda = \{\lambda_1 < \dots < \lambda_k\}$, and for each $i \in \{1,\dots,k\}$ a subtree D_i deriving $\Gamma \vdash M : \lambda_i$. If the sort of M is o, then we take $tr(D) = tr(D_0)$; otherwise, $tr(D) = tr(D_0)\, tr(D_1) \dots tr(D_k)$.

We notice that for $\lambda = (S,\tau)$ the sort of $tr(D)$ is indeed $tr(\tau)$.

We see that arguments of sort o are ignored while transforming an application. Because of that, we need to collect the result of the

$$\frac{}{\Gamma, x:\lambda \vdash x:\lambda} \qquad \frac{}{\Gamma \vdash A : (\emptyset, \tau)} \qquad \frac{}{\Gamma \vdash a^0 : (\{a^0\}, \mathbf{r})} \qquad \frac{r \geq 1}{\Gamma \vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})}$$

$$\frac{\Gamma \vdash L : (S_0, \{(S_1,\tau_1), \ldots, (S_k,\tau_k)\} \to \tau) \qquad \Gamma \vdash M : (S_i, \tau_i) \text{ for each } i \in \{1,\ldots,k\}}{\Gamma \vdash L\, M : (S_0 \cup S_1 \cup \cdots \cup S_k, \tau)} \quad \text{provided that } S_0 \cap (S_1 \cup \cdots \cup S_k) = \emptyset$$

Figure 1. Type system for tracing nullary symbols in a term

$$\frac{\Gamma \vdash b^1 : (\emptyset, \{(e,\mathbf{r})\} \to \mathbf{r}) \quad \dfrac{\Gamma \vdash g : (\emptyset, \{(e,\mathbf{r})\} \to \mathbf{r}) \quad \Gamma \vdash x : (e,\mathbf{r})}{\Gamma \vdash g\, x : (e,\mathbf{r})}}{\Gamma \vdash b^1\, (g\, x) : (e,\mathbf{r})}$$

Figure 2. An example derivation

transformation for all those subtrees of the derivation that describe terms of sort o. This is realized by the tr_{cum} operation that returns a list of terms of sort o. When D is a derivation for a term of sort α, and subtrees of D starting in the children of the root are $D_1, \ldots, D_m$, then

$$tr_{cum}(D) = \begin{cases} tr(D); tr_{cum}(D_1); \ldots; tr_{cum}(D_m) & \text{if } \alpha = o, \\ tr_{cum}(D_1); \ldots; tr_{cum}(D_m) & \text{otherwise.} \end{cases}$$

For a list $R_1; \ldots; R_k$ of terms of sort o, let us define the term $merge(R_1; \ldots; R_k)$ as $\bullet^k\, R_1\, \ldots\, R_k$. Finally, for a substitution η, and a list of terms $list$ we write $list[\eta]$ for the list where the substitution is performed on every term of the $list$.

Example 3. To see an example of such a translation take the term $b^1\, (g\, x)$ that is on the right side of the rule for B in Example 2. For readability of types, we write $(e,\mathbf{r})$ instead of $(\{e^0\},\mathbf{r})$. We take an environment $\Gamma \equiv x : (e,\mathbf{r}), g : (\emptyset, \{(e,\mathbf{r})\} \to \mathbf{r})$ and consider the derivation presented in Figure 2. Calling this derivation D, we have $tr(D) = b^0$ and $tr_{cum}(D) = b^0; g\restriction_{(\emptyset,\{(e,\mathbf{r})\}\to\mathbf{r})}; \bullet^0$. Finally, $merge(tr_{cum}(D)) = \bullet^3\, b^0\, g\restriction_{(\emptyset,\{(e,\mathbf{r})\}\to\mathbf{r})}\, \bullet^0$ is the (slightly perturbed) result of the transformation of the rule for B in Example 2.

The new HORS $\mathcal{S}'$ is created as follows. The rule $A_{init} \to A\, e_1^0\, \ldots\, e_{|\Delta|}^0$ from the initial nonterminal of $\mathcal{S}$ is replaced by $A_{init} \to merge(A_{\tau_0}; e_1^0; \ldots; e_{|\Delta|}^0)$ where $\tau_0 = \{(\{e_1^0\},\mathbf{r})\} \to \cdots \to \{(\{e_{|\Delta|}^0\},\mathbf{r})\} \to \mathbf{r}$. For every other rule of $\mathcal{S}$ of the form $A^\alpha\, x_1^{\alpha_1}\, \ldots\, x_k^{\alpha_k} \to K$ we create a rule in $\mathcal{S}'$ for every derivation of K. More precisely, for each $i \in \{1,\ldots,k\}$ consider the (separated) set of labeled types $\Lambda_i = \{\lambda_{i,1} < \cdots < \lambda_{i,n_i}\}$, where $\lambda_{i,j} = (S_{i,j}, \tau_{i,j})$ for every i,j. For every derivation D of the form $x_1 : \Lambda_1, \ldots, x_k : \Lambda_k \vdash K : (\bigcup_{i\in\{1,\ldots,k\}} \bigcup_{j\in\{1,\ldots,n_i\}} S_{i,j}, \mathbf{r})$ we create a rule

$$A_\tau\, \overline{\mathbf{x_1}}\, \ldots\, \overline{\mathbf{x_k}} \to merge(tr_{cum}(D))\,,$$

where $\tau = (\Lambda_1 \to \cdots \to \Lambda_k \to \mathbf{r}) \in \mathcal{T}^\alpha$, and $\overline{\mathbf{x_i}}$ denotes $x_i\restriction_{\lambda_{i,1}}\, \ldots\, x_i\restriction_{\lambda_{i,n_i}}$ if $\alpha_i \neq o$, and the empty sequence of variables if $\alpha_i = o$ (for $i \in \{1,\ldots,k\}$).

The correctness of the transformation is described by the following two lemmata, which are proved in the next two subsections. Their statements refer to the notion of equivalence introduced at the beginning of this section.

Lemma 5.1 (Soundness)**.** *For every tree generated by $\mathcal{S}'$ there exists an equivalent tree generated by $\mathcal{S}$.*

Lemma 5.2 (Completeness)**.** *For every tree generated by $\mathcal{S}$ there exists an equivalent tree generated by $\mathcal{S}'$.*

5.3 Soundness

To prove Lemma 5.1, we follow a sequence of reductions of $\mathcal{S}'$, and we construct corresponding reductions of $\mathcal{S}$. We however need to assume that the sequence of reductions in $\mathcal{S}'$ is leftmost. We write $P \to^{lf}_{\mathcal{S}'} P'$ to denote that this is the *leftmost* reduction: in $\bullet^k\, P_1\, \ldots\, P_k$ we can reduce inside P_i only when in $P_1, \ldots, P_{i-1}$ there are no more nonterminals. Not surprisingly, the order of reductions does not influence the final result, as stated in the following lemma.

Lemma 5.3. *Suppose that a tree Q can be reached from a term P using some sequence of reductions of $\mathcal{S}'$. Then Q can be reached from P using a sequence of reductions of $\mathcal{S}'$ of the same length in which all reductions are leftmost.*

We need to generalize the definition of equivalence from trees to (lists of) terms of sort o possibly containing nonterminals. We say that two lists of terms of sort o are *merge-equivalent* if one can be obtained from the other by:

- permuting its elements,
- adding or removing the $\bullet^0$ term,
- merging/unmerging some list elements using the symbol $\bullet^k$.

The following property of *merge-equivalent* lists should be clear.

Lemma 5.4. *Let $list$ and $list'$ be two merge-equivalent lists of terms of sort o. Suppose that a tree Q can be generated by $\mathcal{S}'$ from $merge(list)$. Then some tree Q' equivalent to Q can be generated by $\mathcal{S}'$ from $merge(list')$ using a sequence of reductions of the same length.*

The next lemma contains an important observation, needed later in the proof of Lemma 5.6.

Lemma 5.5. *If D derives $\vdash K : (\emptyset, \tau)$, then $tr_{cum}(D)$ is empty.*

Proof. By induction on the structure of D. Recall that $\mathcal{LT}^o$ does not contain pairs with $\emptyset$ on the first coordinate, so the sort of K is not o, and thus $tr_{cum}(D)$ is defined as the concatenation of $tr_{cum}(\cdot)$ for the subtrees of D starting in the children of the root. When D consists of a single node, we immediately have that $tr_{cum}(D)$ is empty. Otherwise $K = L\, M$, and the subtrees of D starting in the children of the root are D_0 deriving $\vdash L : (S_0, \{(S_1,\tau_1), \ldots, (S_k,\tau_k)\} \to \tau)$ and D_i deriving $\vdash M : (S_i, \tau_i)$ for $i \in \{1,\ldots,k\}$. Since $\emptyset = S_0 \cup \cdots \cup S_k$, we have $S_i = \emptyset$ for every $i \in \{0,\ldots,k\}$. The induction assumption implies that $tr_{cum}(D_i)$ is empty for every $i \in \{0,\ldots,k\}$, and thus $tr_{cum}(D)$ is empty. □

Before relating reductions in the HORSes, we analyze what happens during a substitution.

Lemma 5.6. *Consider a derivation D_K for $\Gamma \vdash K^{\alpha_K} : (S,\tau)$. Suppose all bindings for a variable x^{α_x} in Γ are $(x:\lambda_1), \ldots, (x:\lambda_k)$, where $\lambda_1 < \cdots < \lambda_k$, and $(\{\lambda_1, \ldots, \lambda_k\} \to \mathbf{r})$ is a type in $\mathcal{T}^{\alpha_x \to o}$. Suppose also that we have a closed term N^{α_x} with, for every $i = 1,\ldots,k$, a derivation D_i for $\vdash N : \lambda_i$. Then there is a derivation D' for $\Gamma \vdash K[N/x] : (S,\tau)$ such that $tr_{cum}(D')$ is*

merge-equivalent to $tr_{cum}(D_K)[\eta]; tr_{cum}(D_{i_1}); \ldots; tr_{cum}(D_{i_m})$ *where* $\eta = (tr(D_1)/x\restriction_{\lambda_1}, \ldots, tr(D_k)/x\restriction_{\lambda_k})$*, and* $i_1 < \cdots < i_m$ *are those among* $i \in \{1, \ldots, k\}$ *for which in* D_K *there is a node labeled by* $\Gamma \vdash x : \lambda_i$*. Moreover, if* $\alpha_K \neq o$ *then* $tr(D') = tr(D_K)[\eta]$*.*

Proof. Induction on the structure of K. We consider three cases.

The trivial case is when K is a nonterminal, or a symbol, or a variable other than x. Then $K[N/x] = K$, so as D' we can take D_K. Notice that we have $m = 0$ and that the substitution η does not change neither $tr_{cum}(D_K)$ nor $tr(D_K)$ since variables $x\restriction_{\lambda_i}$ do not appear in these terms.

Another easy case is when $K = x$, and thus $(S, \tau) = \lambda_l$ for some $l \in \{1, \ldots, k\}$. We have $m = 1$, and $i_1 = l$, and $K[N/x] = N$. The required derivation D' is obtained from D_l by prepending the type judgment in every its node by the type environment Γ. Clearly D' remains a valid derivation, and $tr(D') = tr(D_l)$ and $tr_{cum}(D') = tr_{cum}(D_l)$. We see that $tr_{cum}(D_K)$ is either the empty list (when $\alpha_K \neq o$) or $\bullet^0$ (when $\alpha_K = o$), so attaching $tr_{cum}(D_K)[\eta]$ does not change the class of *merge*-equivalence. If $\alpha_K \neq o$, we have $tr(D_K)[\eta] = x\restriction_{\lambda_l}[\eta] = tr(D_l)$.

A more involved case is when $K = L^{\alpha_L}\, M^{\alpha_M}$. Then in D_K, below its root, we have a subtree C_0 deriving $\Gamma \vdash L : (S_0, \{(S_1, \tau_1), \ldots, (S_n, \tau_n)\} \to \tau)$, and for each $j \in \{1, \ldots, n\}$ a subtree C_j deriving $\Gamma \vdash M : (S_j, \tau_j)$, where $(S_1, \tau_1) < \cdots < (S_n, \tau_n)$, and $S_0 \cap (S_1 \cup \cdots \cup S_n) = \emptyset$, and $S = S_0 \cup \cdots \cup S_n$. We apply the induction assumption to all these subtrees, obtaining a derivation C_0' for $\Gamma \vdash L[N/x] : (S_0, \{(S_1, \tau_1), \ldots, (S_n, \tau_n)\} \to \tau)$ and for each $j \in \{1, \ldots, n\}$ a derivation C_j' for $\Gamma \vdash M[N/x] : (S_j, \tau_j)$. We compose these derivations into a single derivation D' for $\Gamma \vdash K[N/x] : (S, \tau)$ using the application rule. It remains to prove the required equalities about tr_{cum} and tr.

Let us first see that $tr(D') = tr(D_K)[\eta]$ (not only if $\alpha_K \neq o$, but also if $\alpha_K = o$). From the induction assumption we know that $tr(C_0') = tr(C_0)[\eta]$, as surely $\alpha_L \neq o$. If $\alpha_M = o$, we simply have $tr(D') = tr(C_0')$ and $tr(D_K) = tr(C_0)$, so clearly $tr(D') = tr(D_K)[\eta]$ holds. If $\alpha_M \neq o$, from the induction assumption we also know that $tr(C_j') = tr(C_j)[\eta]$ for every $j \in \{1, \ldots, n\}$; we have $tr(D') = tr(C_0')\, tr(C_1') \ldots tr(C_n')$ and similarly $tr(D_K) = tr(C_0)\, tr(C_1) \ldots tr(C_n)$, so we also obtain $tr(D') = tr(D_K)[\eta]$.

Next, we prove that $tr_{cum}(D')$ is *merge*-equivalent to the list $tr_{cum}(D_K)[\eta]; tr_{cum}(D_{i_1}); \ldots; tr_{cum}(D_{i_m})$. For each $j \in \{0, \ldots, n\}$, let $i_{j,1} < \cdots < i_{j,m_j}$ be those among $i \in \{1, \ldots, k\}$ for which in C_j there is a node labeled by $\Gamma \vdash x : \lambda_i$. By definition $tr_{cum}(D')$ consists of $tr_{cum}(C_j')$ for $j \in \{0, \ldots, n\}$, and if $\alpha_K = o$ then also of $tr(D')$. Similarly, $tr_{cum}(D_K)[\eta]$ consists of $tr_{cum}(C_0)[\eta]; \ldots; tr_{cum}(C_n)[\eta]$, and of $tr(D_K)[\eta]$ if $\alpha_K = o$. We have already shown that $tr(D') = tr(D_K)[\eta]$. The induction assumption implies that $tr_{cum}(C_j')$ is *merge*-equivalent to $tr_{cum}(C_j)[\eta]; tr_{cum}(D_{i_{j,1}}); \ldots; tr_{cum}(D_{i_{j,m_j}})$ for each $j \in \{0, \ldots, n\}$. It remains to observe that the concatenation of the lists $tr_{cum}(D_{i_{j,1}}); \ldots; tr_{cum}(D_{i_{j,m_j}})$ for $j \in \{0, \ldots, n\}$ is *merge*-equivalent to $tr_{cum}(D_{i_1}); \ldots; tr_{cum}(D_{i_m})$. By definition every $i_{j,l}$ equals to some $i_{l'}$ and every i_l equals to some $i_{j,l'}$; the only question is about duplicates on these lists. Let us write $\lambda_i = (T_i, \sigma_i)$ for every $i \in \{1, \ldots, k\}$. When some i_l is such that $T_{i_l} = \emptyset$, then the list $tr_{cum}(D_{i_l})$ is empty (Lemma 5.5), so anyway we do not have to care about duplicates. On the other hand, when $T_{i_l} \neq \emptyset$ and a node labeled by $\Gamma \vdash x : \lambda_{i_l}$ appears in some C_j, then $T_{i_l} \subseteq S_j$. Since the sets $S_0, \ldots, S_n$ are disjoint, such node appears in C_j only for one j, and thus such i_l equals to only one among the $i_{j,l'}$'s. □

We can now formulate and prove the key lemma of this section, allowing us to simulate a single step of $\mathcal{S}'$ by a single step of $\mathcal{S}$.

Lemma 5.7. *Let* D *be a derivation for* $\vdash L : (S, \mathbf{r})$*, where* L *does not contain the initial nonterminal of* $\mathcal{S}$*. If* $merge(tr_{cum}(D)) \to^{lf}_{\mathcal{S}'} P$*, then there exists a term* L' *and a derivation* D' *for* $\vdash L' : (S, \mathbf{r})$ *such that* $L \to_{\mathcal{S}} L'$ *and* $tr_{cum}(D')$ *is merge-equivalent to* P*.*

Proof. We proceed by induction on the structure of L.

Suppose first that $L = a^r\, M_1 \ldots M_r$ (where surely $r \geq 1$). Then D starts with a sequence of r application rules followed by a single-node derivation for $\vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})$, and by derivations D_i for $\vdash M_i : (S_i, \mathbf{r})$, for each $i \in \{1, \ldots, r\}$. In particular, $S_1, \ldots, S_r$ are disjoint and their union is S. It holds that $tr_{cum}(D) = (a^0; tr_{cum}(D_1); \ldots; tr_{cum}(D_r))$. The reduction $merge(tr_{cum}(D)) \to^{lf}_{\mathcal{S}'} P$ concerns one of terms on one of the lists $tr_{cum}(D_i)$, and thus we can write $P = merge(a^0; list_1'; \ldots; list_r')$, where for some $l \in \{1, \ldots, r\}$ we have $merge(tr_{cum}(D_l)) \to^{lf}_{\mathcal{S}'} merge(list_l')$, and $tr_{cum}(D_i) = list_i'$ for $i \neq l$. We apply the induction assumption to M_l, obtaining a term M_l' and a derivation D_l' for $\vdash M_l' : (S_l, \mathbf{r})$ such that $M_l \to_{\mathcal{S}} M_l'$ and that $tr_{cum}(D_l')$ is *merge*-equivalent to $list_l'$. Taking $D_i' = D_i$ and $M_i' = M_i$ for $i \neq l$, and $L' = a^r\, M_1' \ldots M_r'$, we have $L \to_{\mathcal{S}} L'$. Out of a node labeled by $\vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})$ and of derivations D_i' for $i \in \{1, \ldots, r\}$ we compose a derivation D', using the application rule r times. We have $tr_{cum}(D') = (a^0; tr_{cum}(D_1'); \ldots; tr_{cum}(D_r'))$, and thus $tr_{cum}(D')$ is *merge*-equivalent to P.

The remaining possibility is that $L = A\, N_1^{\alpha_1} \ldots N_k^{\alpha_k}$. Then D starts with a sequence of application rules ending in a single-node derivation for $\vdash A : (\emptyset, \tau)$ with $\tau = \{\lambda_{1,1}, \ldots, \lambda_{1,n_1}\} \to \ldots, \to \{\lambda_{k,1}, \ldots, \lambda_{k,n_k}\} \to \mathbf{r}$, and in derivations $D_{i,j}$ for $\vdash N_i : \lambda_{i,j}$, for each $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n_i\}$. Suppose $\lambda_{i,1} < \cdots < \lambda_{i,n_i}$ for every $i \in \{1, \ldots, k\}$, and $\lambda_{i,j} = (S_{i,j}, \tau_{i,j})$ for every i, j. Since we consider the leftmost reduction of $merge(tr_{cum}(D))$, it necessarily concerns its part $tr(D)$ (which is the first term in the list $tr_{cum}(D)$), that consists of the nonterminal $A\restriction_\tau$ to which some of the terms $tr(D_{i,j})$ are applied (namely, terms $tr(D_{i,j})$ for those i for which $\alpha_i \neq o$). This reduction uses some rule $A_\tau\, \overline{\mathbf{x_1}} \ldots \overline{\mathbf{x_k}} \to merge(tr_{cum}(D_K))$, where in $\mathcal{S}$ we have a rule $A\, x_1 \ldots x_k \to K$, and we have a derivation D_K for $\Gamma \vdash K : (S, \tau)$ with $\Gamma = \bigcup_{i \in \{1, \ldots, k\}} \bigcup_{j \in \{1, \ldots, n_i\}} \{x_i : \lambda_{i,j}\}$, and where $\overline{\mathbf{x_i}}$ denotes $x_i\restriction_{\lambda_{i,1}} \ldots x_i\restriction_{\lambda_{i,n_i}}$ if $\alpha_i \neq o$ and the empty sequence of variables if $\alpha_i = o$ (for $i \in \{1, \ldots, k\}$).

As L' we take the result of applying the rule $A\, x_1 \ldots x_k \to K$ to L, i.e. $L' = K[N_1/x_1, \ldots, N_k/x_k]$. To construct a derivation for it, we construct derivations $D_{i,K}$ for $K[N_1/x_1, \ldots, N/x_i]$, for $i = 1, \ldots, k$. We take $D_{0,K} = D_K$. To obtain $D_{i,K}$ we apply Lemma 5.6 to N_i, $D_{i-1,K}$ and $D_{i,1}, \ldots, D_{i,n_i}$. The derivation $D_{k,K}$ derives $\Gamma \vdash L' : (S, \mathbf{r})$. Let D' be the derivation for $\vdash L' : (S, \mathbf{r})$ obtained from $D_{k,K}$ by removing the type environment Γ from type judgments in all its nodes; we obtain a valid derivation since L' is closed.

It remains to see that $tr_{cum}(D')$ is *merge*-equivalent to P. Let *list* be the concatenation of lists $tr_{cum}(D_{i,j})$ for all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n_i\}$ and let $Q = merge(tr_{cum}(D_K))[\eta_1, \ldots, \eta_k]$ where $\eta_i = (tr(D_{i,1})/x_i\restriction_{\lambda_{i,1}}, \ldots, tr(D_{i,n_i})/x_i\restriction_{\lambda_{i,n_i}})$ for $i \in \{1, \ldots, k\}$; we see that Q is the result of applying the considered rule to $tr(D)$ (substitutions η_i for i such that $\alpha_i = o$ can be skipped, since anyway variables $x_i\restriction_{\lambda_{i,j}}$ for such i do not appear in $tr(D_K)$). For $i \in \{1, \ldots, k\}$, let $j_{i,1} < \cdots < j_{i,m_i}$ be those among $j \in \{1, \ldots, n_i\}$ for which in D_K there is a node labeled by $\Gamma \vdash x : \lambda_{i,j}$. By definition $tr_{cum}(D) = (tr(D); list)$, and thus $P = merge(Q; list)$. On the other hand Lemma 5.6 says that $tr_{cum}(D')$ is *merge*-equivalent to $Q; list'$, where $list'$ is the concatenation of $tr_{cum}(D_{i,j_1}); \ldots; tr_{cum}(D_{i,j_{m_i}})$ for $i \in \{1, \ldots, k\}$. We notice, however, that $list = list'$. Indeed, if

$S_{i,j} = \emptyset$ for some i, j, then $tr_{cum}(D_{i,j})$ is empty by Lemma 5.5. Suppose that $S_{i,j} \neq \emptyset$. The rules of the type system ensure that the subset of Δ in the root of D_K (that is S) is the union of those subsets in all leaves of D_K. We have assumed that symbols from Δ do not appear in K (they are allowed to appear only in the rule from the initial nonterminal). Moreover, $S_{i,j} \subseteq S$, and all other sets $S_{i',j'}$ are disjoint from $S_{i,j}$ (by the definition of types). Thus necessarily a node labeled by $\Gamma \vdash x_i : \lambda_{i,j}$ appears in D_K (this is the only way the elements of $S_{i,j}$ can be introduced in S). This means that our j is listed among $j_{i,1}, \ldots, j_{i,m_i}$, and hence $tr_{cum}(D_{i,j})$ appears in $list'$. This proves that $list = list'$, and in consequence that $tr_{cum}(D')$ is $merge$-equivalent to P. □

Lemma 5.8. *Let D be a derivation for $\vdash L : (S, \mathbf{r})$ such that $merge(tr_{cum}(D))$ is a tree. Then L is a tree, and is equivalent to $merge(tr_{cum}(D))$.*

Proof. Induction on the structure of L. If L was of the form $A\, M_1 \ldots M_k$, then in D we would necessarily have a node for the nonterminal A, which would imply that $merge(tr_{cum}(D))$ is not a tree, i.e., it contains a nonterminal. Thus L is of the form $a^r\, M_1 \ldots M_r$. Looking at the type system we notice that D necessarily starts with a sequence of r application rules followed by a single-node derivation for $\vdash a^r : (S_0, \tau_0)$, and derivations D_i for $\vdash M_i : (S_i, \mathbf{r})$, for $i \in \{1, \ldots, r\}$. Recall that $tr_{cum}(D) = (a^0; tr_{cum}(D_1); \ldots; tr_{cum}(D_r))$. For $i \in \{1, \ldots, r\}$ we know that $merge(tr_{cum}(D_i))$ is a tree; the induction assumption implies that M_i is a tree, and is equivalent to $merge(tr_{cum}(D_i))$. It follows that L is a tree, and is equivalent to $merge(tr_{cum}(D))$. □

Corollary 5.9. *Let D be a derivation for $\vdash L : (S, \mathbf{r})$, where L does not contain the initial nonterminal. If a tree Q can be generated by $\mathcal{S}'$ from $merge(tr_{cum}(D))$, then a tree equivalent to Q can be generated by $\mathcal{S}$ from L.*

Proof. Induction on the smallest length of a sequence of reductions $merge(tr_{cum}(D)) \to^*_{\mathcal{S}'} Q$. If this length is 0, we apply Lemma 5.8. Suppose that the length is positive. Thanks to Lemma 5.3 we can write $merge(tr_{cum}(D)) \to^{lf}_{\mathcal{S}'} P \to^*_{\mathcal{S}'} Q$ (without changing the length of the sequence of reductions). Using Lemma 5.7 we obtain a term L' and a derivation D' for $\vdash L' : (S, \mathbf{r})$ such that $L \to_{\mathcal{S}} L'$ and that $tr_{cum}(D')$ is $merge$-equivalent to P. The initial nonterminal does not appear in L' since by assumption it does not appear on the right side of any rule. Because $P \to^*_{\mathcal{S}'} Q$, by Lemma 5.4 we also have a sequence of reductions of the same length $merge(tr_{cum}(D')) \to^*_{\mathcal{S}'} Q'$ to some tree Q' equivalent to Q; to this sequence of reductions we apply the induction assumption. □

Proof of Lemma 5.1. Let $n = |\Delta|$. Consider the rule $A_{init} \to A\, e^0_1 \ldots e^0_n$ from the initial nonterminal of $\mathcal{S}$. Let D be a derivation for $\vdash A\, e^0_1 \ldots e^0_n : (\Delta, \mathbf{r})$ that consists of a node labeled by $\vdash A : (\emptyset, \tau)$ with $\tau = \{(\{e^0_1\}, \mathbf{r})\} \to \ldots \to \{(\{e^0_n\}, \mathbf{r})\} \to \mathbf{r}$, and of nodes labeled by $\vdash e^0_i : (\{e^0_i\}, \mathbf{r})$ for $i \in \{1, \ldots, n\}$, joined together by application rules. We see that $tr_{cum}(D) = (A_\tau; e^0_1; \ldots; e^0_n)$.

Take a tree Q generated by $\mathcal{S}'$. Since the only rule of $\mathcal{S}'$ from the initial nonterminal is $A_{init} \to merge(A_\tau; e^0_1; \ldots; e^0_n)$, the tree Q is generated by $\mathcal{S}'$ also from $merge(tr_{cum}(D))$. By Corollary 5.9 a tree Q' equivalent to Q can be generated by $\mathcal{S}$ from $A\, e^0_1 \ldots e^0_n$, and thus also from the initial nonterminal. □

5.4 Completeness

The proof of Lemma 5.2 is similar to the one of Lemma 5.1; we just need to proceed in the opposite direction. Namely, we take a sequence of reductions of $\mathcal{S}$ finishing in a finite tree, and then working from the end of the sequence we construct backwards a sequence of reductions of $\mathcal{S}'$.

There is one additional difficulty that was absent in the previous subsection: we need some kind of uniqueness of derivations. Indeed, while proceeding forwards from $A\, N_1 \ldots N_k$ to $K[N_1/x_1, \ldots, N_k/x_k]$, we take a derivation for N_1 from the single place where N_1 appears in the first term, and we put it in multiple places where N_1 appears in the second term. This time we proceed backwards, so there are multiple places in the second term where we have a derivation for N_1. Our type system can accommodate different derivations for the occurrences of N_1 having different types, but for each type we have to ensure that in different occurrences of N_1 with this type the derivations are the same. Because of that we only consider maximal derivations.

A derivation D is called *maximal* if for every internal node of D the following holds: if the label of this node is $\Gamma \vdash L\, M : (S, \tau)$ and it is possible to derive $\Gamma \vdash M : (\emptyset, \sigma)$ for some σ, then necessarily this node has a child labeled by $\Gamma \vdash M : (\emptyset, \sigma)$. The following two lemmata say that it is enough to consider only maximal derivations, and that maximal derivations are unique if we restrict ourselves to labeled types with empty subset of Δ. We will see later that for other types the multiple occurrence problem mentioned above does not occur.

Lemma 5.10. *If $\vdash K : (S, \tau)$ can be derived, then it can be derived by a maximal derivation.*

Proof. Let $\tau = \Lambda_1 \to \ldots \to \Lambda_n \to \mathbf{r}$ and suppose D is a derivation for $\vdash K^\alpha : (S, \tau)$. We prove a stronger statement: if $T_1, \ldots, T_n$ are such that $\tau' = ((\Lambda_1 \cup (\{\emptyset\} \times T_1)) \to \ldots \to (\Lambda_n \cup (\{\emptyset\} \times T_n)) \to \mathbf{r})$ is a type in $\mathcal{T}^\alpha$ then there exists a maximal derivation D' for $\vdash K : (S, \tau')$. This is shown by induction on the structure of K. Surely K is not a variable, as then a type judgment with empty type environment could not be derived. If K is a nonterminal, then $S = \emptyset$, and $\vdash K : (S, \tau')$ (for any $\tau' \in \mathcal{T}^\alpha$) can be derived by a single-node derivation; this is a maximal derivation. If K is a symbol, its sort is $o^n \to o$; by definition of $\mathcal{LT}^o$ we know that $T_i = \emptyset$ for every $i \in \{1, \ldots, n\}$, which implies $\tau' = \tau$. Thus D derives $\vdash K : (S, \tau')$ and is maximal, since it consists of a single node.

Finally, suppose that $K = L\, M$. Then in D we have a subtree D_i deriving $\vdash L : (S_0, \Lambda_0 \to \tau)$, and for every $\lambda \in \Lambda_0$ a subtree D_λ deriving $\vdash M : \lambda$. Let T_0 contain those σ for which we can derive $\vdash M : (\emptyset, \sigma)$ but $(\emptyset, \sigma) \notin \Lambda_0$. Then by the induction assumption there exists a maximal derivation D'_0 for $\vdash L : (S_0, (\Lambda_0 \cup (\{\emptyset\} \times T_0)) \to \tau')$, and for every $\lambda \in (\Lambda_0 \cup (\{\emptyset\} \times T_0))$ there exists a maximal derivation D'_λ for $\vdash M : \lambda$. By composing these derivations together, we obtain a maximal derivation D' for $\vdash K : (S, \tau')$: the side condition of the application rule still holds since we have added only derivations for labeled types of the form $(\emptyset, \sigma)$. □

Lemma 5.11. *For every type judgment of the form $\Gamma \vdash K : (\emptyset, \tau)$ there exists at most one maximal derivation D deriving it.*

Proof. By induction on the structure of K. If K is a variable, a symbol, or a nonterminal, then D necessarily consists of a single node labeled by the resulting type judgment, so it is unique. Suppose that $K = L\, M$. Then below the root of D, labeled by $\Gamma \vdash K : (\emptyset, \tau)$, we have a subtree D_0 deriving $\Gamma \vdash L : (\emptyset, \{\emptyset\} \times T \to \tau)$, and for every $\sigma \in T$ a subtree D_σ deriving $\Gamma \vdash M : (\emptyset, \sigma)$. By maximality, whenever we can derive $\Gamma \vdash M : (\emptyset, \sigma)$ for some σ, there should be a child of the root of D labeled by $\Gamma \vdash M : (\emptyset, \sigma)$, and then $\sigma \in T$. This fixes the set T, and thus the set of child labels. The derivations D_0 and D_τ for $\tau \in T$ are unique by the induction assumption. □

After these preparatory results about derivations we come back to our proof. The next lemma deals with the base case: for the last term in a sequence of reductions in $\mathcal{S}$ (this term is a narrow tree) we create an equivalent term that will be the last term in the corresponding sequence of reductions in $\mathcal{S}'$.

Lemma 5.12. *Let $S \subseteq \Delta$, and let K be an S-narrow tree. Then there exists a maximal derivation D for $\vdash K : (S, \mathbf{r})$ such that $merge(tr_{cum}(D))$ is a tree equivalent to K.*

Proof. We proceed by induction on the structure of K, which is necessarily of the form $a^r\ M_1\ \ldots\ M_r$. If $r = 0$, then $S = \{a^0\}$ and we take D to be the single-node derivation for $\vdash a^0 : (\{a^0\}, \mathbf{r})$; we have $tr_{cum}(D) = a^0$. Suppose that $r \geq 1$. Then S can be represented as a union of disjoint sets $S_1, \ldots, S_r$ s.t. M_i is a S_i-narrow tree for each $i \in \{1, \ldots, r\}$. By induction, $\forall i \in \{1, \ldots, r\}$ we obtain a maximal derivation D_i for $\vdash M_i : (S_i, \mathbf{r})$ s.t. $merge(tr_{cum}(D_i))$ is a tree equivalent to M_i. The derivation D is obtained by deriving $\vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \ldots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})$ and attaching $D_1, \ldots, D_r$ using the application rule r times. Because the M_i's are of sort o, and $\mathcal{LT}^o$ does not contain pairs of the form $(\emptyset, \sigma)$, the definition of maximality requires no additional children for the new internal nodes of D, and hence D is maximal. Thus $tr_{cum}(D) = (a^0; tr_{cum}(D_1); \ldots; tr_{cum}(D_r))$. □

We now describe what happens during a substitution.

Lemma 5.13. *Suppose that D' is a maximal derivation for $\Gamma \vdash K^{\alpha_K}[N/x^{\alpha_x}] : (S, \tau)$, where N is closed. Let $\Lambda_\emptyset$ be the set of those $(\emptyset, \sigma) \in \mathcal{LT}^{\alpha_x}$ for which $\vdash N : (\emptyset, \sigma)$ can be derived. Then there exists a set $\Lambda \in \mathcal{LT}^{\alpha_x}$, a maximal derivation D_K for $\Gamma' \vdash K : (S, \tau)$ with $\Gamma' = \Gamma \cup \{x : \lambda \mid \lambda \in \Lambda\}$, and for each $\lambda \in \Lambda$ a maximal derivation D_λ for $\vdash N : \lambda$, such that*

1. *$\Lambda_\emptyset \subseteq \Lambda$,*
2. *for every $\lambda \in \Lambda \setminus \Lambda_\emptyset$ in D_K there is a node labeled by $\Gamma' \vdash x : \lambda$,*
3. *the list $tr_{cum}(D')$ is $merge$-equivalent to the list $tr_{cum}(D_K)[\eta]; tr_{cum}(D_{\lambda_1}); \ldots; tr_{cum}(D_{\lambda_k})$, where $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$ with $\lambda_1 < \cdots < \lambda_k$, and $\eta = (tr(D_{\lambda_1})/x\restriction_{\lambda_1}, \ldots, tr(D_{\lambda_k})/x\restriction_{\lambda_k})$, and*
4. *if $\alpha_K \neq o$ then also $tr(D') = tr(D_K)[\eta]$.*

Proof. We proceed by induction on the structure of K. By Lemma 5.10, for $\lambda \in \Lambda_\emptyset$, there exists a maximal derivation for $\vdash N : \lambda$, which is unique by Lemma 5.11. We denote this unique derivation by D_λ.

We consider three cases. First suppose that K is a nonterminal, or a symbol, or a variable other than x. In this case $K[N/x] = K$. We take $\Lambda = \Lambda_\emptyset$, and to obtain D_K we just extend the type environment in the only node of D' by $\{x : \lambda \mid \lambda \in \Lambda\}$. Points 1-2 hold trivially. For points 3-4 we observe that neither $tr(D_K)$ nor $tr_{cum}(D_K)$ contains a variable $x\restriction_\lambda$ (so the substitution η does not change these terms); additionally $tr_{cum}(D_\lambda)$ for $\lambda \in \Lambda$ are empty (Lemma 5.5).

Next, suppose that $K = x$. We take $\Lambda = \Lambda_\emptyset \cup \{(S, \tau)\}$. As D_K we take the single-node derivation for $\Gamma' \vdash x : (S, \tau)$, and as $D_{(S,\tau)}$ we take D' in which we remove the type environment from every node. Since N is closed, $D_{(S,\tau)}$ remains a valid derivation and it remains maximal (when $(S, \tau) \in \Lambda_\emptyset$, we have already defined $D_{(S,\tau)}$ previously, but these two definitions give the same derivation). Points 1-2 hold trivially. We have $tr(D') = tr(D_{(S,\tau)})$ and $tr_{cum}(D') = tr_{cum}(D_{(S,\tau)})$. We see that $tr_{cum}(D_K)$ is either an empty list (when $\alpha_K \neq o$) or $\bullet^0$ (when $\alpha_K = o$), so attaching $tr_{cum}(D_K)[\eta]$ does not change the class of $merge$-equivalence. Moreover $tr_{cum}(D_\lambda)$ for $\lambda \in \Lambda_\emptyset$ are empty (Lemma 5.5), which gives point 3. If $\alpha_K \neq o$, we have $tr(D_K)[\eta] = x\restriction_{(S,\tau)}[\eta] = tr(D_{(S,\tau)}) = tr(D')$ (point 4).

Finally suppose that $K = L^{\alpha_L}\ M^{\alpha_M}$, which is a more involved case. In D', below its root, we have a subtree C'_0 deriving $\Gamma \vdash L[N/x] : (S_0, \{(S_1, \tau_1), \ldots, (S_n, \tau_n)\} \to \tau)$, and for each $j \in \{1, \ldots, n\}$ a subtree C'_j deriving $\Gamma \vdash M[N/x] : (S_j, \tau_j)$, where $(S_1, \tau_1) < \cdots < (S_n, \tau_n)$, and $S_0 \cap (S_1 \cup \cdots \cup S_n) = \emptyset$, and $S = S_0 \cup \cdots \cup S_n$. We apply the induction assumption to all these subtrees, obtaining a maximal derivation C_0 for $\Gamma \cup \{x : \lambda \mid \lambda \in \Lambda_0\} \vdash L : (S_0, \{(S_1, \tau_1), \ldots, (S_n, \tau_n)\} \to \tau)$ and for each $j \in \{1, \ldots, n\}$ a maximal derivation C_j for $\Gamma \cup \{x : \lambda \mid \lambda \in \Lambda_j\} \vdash M : (S_j, \tau_j)$, and for each $j \in \{0, \ldots, n\}$ and $\lambda \in \Lambda_j$ a maximal derivation $D_{j,\lambda}$ for $\vdash N : \lambda$.

Let $\Lambda = \bigcup_{j \in \{0,\ldots,n\}} \Lambda_j$. For $\lambda \in \Lambda_\emptyset$ we have already defined D_λ, and we have $D_\lambda = D_{j,\lambda}$ for every $j \in \{0, \ldots, n\}$. Recall that for every $\lambda \in \Lambda_j \setminus \Lambda_\emptyset$ there is a node in C_j deriving the labeled type λ, and hence the set on the first coordinate of λ is a subset of S_j (point 2). Since the sets S_j are disjoint, for every $\lambda \in \Lambda \setminus \Lambda_\emptyset$ there is exactly one j for which $\lambda \in \Lambda_j$, and we define D_λ to be $D_{j,\lambda}$ for this j.

We extend the type environment in every node of every C_j to $\Gamma' = \Gamma \cup \{x : \lambda \mid \lambda \in \Lambda\}$, and we compose these derivations into a single derivation D_K for $\Gamma' \vdash K : (S, \tau)$ using the rule for application. In order to see that D_K is maximal, take some internal node of D_K. Suppose first that this node is contained inside some C_j and it is labeled by $\Gamma' \vdash P\,Q$, and it is possible to derive $\Gamma' \vdash Q : (\emptyset, \sigma)$. Then it is as well possible to derive $\Gamma \cup \{x : \lambda \mid \lambda \in \Lambda_j\} \vdash Q : (\emptyset, \sigma)$, because $\Lambda \setminus \Lambda_j$ contains only labeled types with nonempty set on the first coordinate and they anyway cannot be used while deriving a labeled type with empty set on the first coordinate. Thus by maximality of C_j our node has a child labeled by $\Gamma' \vdash Q : (\emptyset, \sigma)$. Next, consider the root of D_K, and suppose that it is possible to derive $\Gamma' \vdash M : (\emptyset, \sigma)$. Then by Lemma 5.6 it is as well possible to derive $\Gamma' \vdash M[N/x] : (\emptyset, \sigma)$, so also $\Gamma \vdash M[N/x] : (\emptyset, \sigma)$ (since x does not appear in $M[N/x]$), which by maximality of D' means that $(\emptyset, \sigma)$ is one of (S_j, τ_j), and thus the root of D_K has a child labeled by $\Gamma' \vdash M : (\emptyset, \sigma)$ (created out of the root of C_j).

Points 1, 2 follow from the induction assumption. It remains to prove points 3, 4. Let $\Lambda = \{\lambda_1 < \cdots < \lambda_k\}$ and $\eta = (tr(D_{\lambda_1})/x\restriction_{\lambda_1}, \ldots, tr(D_{\lambda_k})/x\restriction_{\lambda_k})$. Similarly, let $\Lambda_j = \{\lambda_{j,1} < \cdots < \lambda_{j,k_j}\}$ and $\eta_j = (tr(D_{\lambda_{j,1}})/x\restriction_{\lambda_{j,1}}, \ldots, tr(D_{\lambda_{j,k}})/x\restriction_{\lambda_{j,k_j}})$. Let us first see that $tr(D') = tr(D_K)[\eta]$ (not only if $\alpha_K \neq o$, as in point 4, but also if $\alpha_K = o$). By induction we know that $tr(C'_0) = tr(C_0)[\eta_0]$, as surely $\alpha_L \neq o$. Thus $tr(C'_0) = tr(C_0)[\eta]$, since $tr(C'_0)$ (hence also $tr(C_0)[\eta_0]$) does not contain variables $x\restriction_\lambda$, so substituting for them does not change anything. If $\alpha_M = o$, we simply have $tr(D') = tr(C'_0)$ and $tr(D_K) = tr(C_0)$, so clearly $tr(D') = tr(D_K)[\eta]$ holds. If $\alpha_M \neq o$, by induction we also know that $tr(C'_j) = tr(C_j)[\eta_j]$ $\forall j \in \{1, \ldots, n\}$, and thus also $tr(C'_j) = tr(C_j)[\eta]$; we have $tr(D') = tr(C'_0)\ tr(C'_1)\ \ldots\ tr(C'_n)$ and similarly $tr(D_K) = tr(C_0)\ tr(C_1)\ \ldots\ tr(C_n)$, so we also obtain $tr(D') = tr(D_K)[\eta]$. To show point 3 we prove that $tr_{cum}(D')$ is $merge$-equivalent to the list $tr_{cum}(D_K)[\eta]; tr_{cum}(D_{\lambda_1}); \ldots; tr_{cum}(D_{\lambda_k})$. By definition $tr_{cum}(D')$ consists of $tr_{cum}(C'_j)$ for $j \in \{0, \ldots, n\}$, and if $\alpha_K = o$ then also of $tr(D')$. Similarly, $tr_{cum}(D_K)[\eta]$ equals to $tr_{cum}(C_0)[\eta]; \ldots; tr_{cum}(C_n)[\eta]$, prepended by $tr(D_K)[\eta]$ if $\alpha_K = o$. We have already shown that $tr(D') = tr(D_K)[\eta]$. By the induction assumption, the list $tr_{cum}(C'_j)$ is $merge$-equivalent to the list $tr_{cum}(C_j)[\eta_j]; tr_{cum}(D_{\lambda_{j,1}}); \ldots; tr_{cum}(D_{\lambda_{j,k_j}})$ for all $j \in \{0, \ldots, n\}$. We can replace here η_j by η, since $tr_{cum}(C'_j)$ does not contain variables $x\restriction_\lambda$ with $\lambda \in \Lambda \setminus \Lambda_j$. To finish the proof it is enough to observe that the concatenation of the lists $tr_{cum}(D_{\lambda_{j,1}}); \ldots; tr_{cum}(D_{\lambda_{j,k_j}})$ for $j \in \{0, \ldots, n\}$ is $merge$-equivalent to $tr_{cum}(D_{\lambda_1}); \ldots; tr_{cum}(D_{\lambda_k})$. Indeed, for $\lambda \in \Lambda_\emptyset$ by Lemma 5.5 $tr_{cum}(D_\lambda)$ is empty, and, as we have already shown, every $\lambda \in \Lambda \setminus \Lambda_\emptyset$ belongs to exactly one Λ_j. □

Lemma 5.14. *Let D' be a maximal derivation for $\vdash L' : (S, \mathbf{r})$, and let L be a term that does not contain the initial nonterminal of $\mathcal{S}$ and such that $L \to_{\mathcal{S}} L'$. Then there exists a maximal derivation*

D *for* $\vdash L : (S, \mathbf{r})$ *and a term* P *that is* $merge$*-equivalent to* $tr_{cum}(D')$ *and such that* $merge(tr_{cum}(D)) \to_{\mathcal{S}'} P$.

The lemma is proved by induction on the structure of L; cf. App. B. The case when L starts with a nonterminal uses Lemma 5.13.

Corollary 5.15. *Let* L *be a term that is of sort* o *and does not contain the initial nonterminal of* $\mathcal{S}$*, and let* M *be an* S*-narrow tree generated by* $\mathcal{S}$ *from* L*. Then there exists a maximal derivation* D *for* $\vdash L : (S, \mathbf{r})$ *such that a tree equivalent to* M *can be generated by* $\mathcal{S}'$ *from* $merge(tr_{cum}(D))$.

Proof. We proceed by induction on the smallest length of the sequence of reductions $L \to_{\mathcal{S}}^* M$. If $L = M$, we just apply Lemma 5.12. Suppose that the length is positive, and write $L \to_{\mathcal{S}} L' \to_{\mathcal{S}}^* M$. The initial nonterminal does not appear in L' since by assumption it does not appear on the right side of any rule. By induction we obtain a maximal derivation D' for $\vdash L' : (S, \mathbf{r})$ such that a tree Q equivalent to M can be generated by $\mathcal{S}'$ from $merge(tr_{cum}(D'))$. Then, from Lemma 5.14 we obtain a maximal derivation D for $\vdash L : (S, \mathbf{r})$ and a term P that is $merge$-equivalent to $tr_{cum}(D')$ and such that $merge(tr_{cum}(D)) \to_{\mathcal{S}'} P$. By Lemma 5.4 a tree equivalent to Q (and hence to M) can be generated by $\mathcal{S}'$ from P, and hence also from $merge(tr_{cum}(D))$. □

Proof of Lemma 5.2. Consider a tree M generated by $\mathcal{S}$, and a sequence of reductions of $\mathcal{S}$ leading to M. In the first step the initial nonterminal reduces to $A\, e_1^0 \ldots e_{|\Delta|}^0$. Corollary 5.15 gives us a derivation D for $\vdash A\, e_1^0 \ldots e_{|\Delta|}^0 : (\Delta, \mathbf{r})$ such that $merge(tr_{cum}(D))$ generates a tree equivalent to M. Necessarily $tr_{cum}(D) = (A_{\tau_0}; e_1^0; \ldots; e_{|\Delta|}^0)$, so $merge(tr_{cum}(D))$ is obtained as the result of the initial rule of $\mathcal{S}'$. □

6. Conclusions

This work leaves open the question of the exact complexity of the diagonal problem. The only known lower bound is given by the emptiness problem, that is the same as for the model-checking problem [21]. Our procedure is probably not optimal, one of the reasons being the use of reflection in operation Theorem 2.1.

References

[1] P. A. Abdulla, L. Boasson, and A. Bouajjani. Effective lossy queue languages. In *In Proc. of ICALP'01*, LNCS, pages 639–651, 2001.

[2] A. V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, Oct. 1968.

[3] K. Asada and N. Kobayashi. On word and frontier languages of unsafe higher-order grammars. To appear in Proc. of ICALP'16.

[4] G. Bachmeier, M. Luttenberger, and M. Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In *In Proc. of LATA'15*, volume 8977 of *LNCS*, pages 473–485, 2015.

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

[6] C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS'10*, pages 120–129, 2010.

[7] L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. Ordered tree-pushdown systems. In *In Proc. of FSTTCS'15*, volume 45 of *LIPIcs*, pages 163–177, 2015.

[8] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, 2007.

[9] B. Courcelle. On constructing obstruction sets of words. *Bulletin of EATCS*, 1991.

[10] W. Czerwiński, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In *FCT'15*, volume 9210 of *LNCS*, pages 173–185, 2015.

[11] W. Czerwiński, W. Martens, L. van Rooijen, M. Zeitoun, and G. Zetzsche. A characterization for decidable separability by piecewise testable languages. Submitted, 2015.

[12] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[13] H. Gruber, M. Holzer, and M. Kutrib. The size of Higman–Haines sets. *Theor. Comput. Sci.*, 387(2):167–176, 2007.

[14] P. Habermehl, R. Meyer, and H. Wimmel. The downward-closure of Petri net languages. In *In Proc. of ICALP'10*, volume 6199 of *LNCS*, pages 466–477, 2010.

[15] M. Hague. Parameterised pushdown systems with non-atomic writes. In *FSTTCS*, volume 13 of *LIPIcs*, pages 457–468, 2011.

[16] M. Hague, J. Kochems, and C.-H. L. Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proc. of POPL'16*, pages 151–163, 2016.

[17] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. of LICS'08*, pages 452–461. IEEE Computer Society, 2008.

[18] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, s3-2(1):326–336, Jan. 1952.

[19] P. Hofman and W. Martens. Separability by short subsequences and subwords. In *ICDT 2015*, volume 31 of *LIPIcs*, pages 230–246, 2015.

[20] N. Kobayashi, K. Inaba, and T. Tsukada. Unsafe order-2 tree languages are context-sensitive. In *In Proc. of FOSSACS'14*, pages 149–163, 2014.

[21] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

[22] G. M. Kobele and S. Salvati. The IO and OI hierarchies revisited. *Information and Computation*, 243:205–221, 2015.

[23] R. Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, Mar. 2003.

[24] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. of LICS'06*, pages 81–90, 2006.

[25] S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. *Logical Methods In Computer Science*, 2015.

[26] S. L. Torre, A. Muscholl, and I. Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *In Proc. of CONCUR'15*, volume 42 of *LIPIcs*, pages 72–84, 2015.

[27] J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Math.*, 21(3):237–252, May 1978.

[28] G. Zetzsche. An approach to computing downward closures. In *In Proc. of ICALP'15*, volume 9135 of *LNCS*, pages 440–451, 2015.

[29] G. Zetzsche. Computing downward closures for stacked counter automata. In *In Proc. of STACS'15*, volume 30 of *LIPIcs*, pages 743–756, 2015.

A. Closure under linear transductions and full trio

In this section we prove that finite tree languages generated by HORSes are closed under *linear* bottom-up tree transductions.

An FTT is *complete* if every variable x_i appearing on the left side of any transition also appears in the term t on the right side of the transition, i.e., no subtree is discarded. A *restriction* is a special case of an FTT where there is only one control state, and where every transition is of the form $a^r\,(q,x_1)\,\ldots\,(q,x_r) \longrightarrow q, b^n\,x_{i_1}\,\ldots\,x_{i_n}$ with $1 \leq i_1 < \cdots < i_n \leq r$, i.e., it relabels the tree and discards some its subtrees. Clearly, every FTT is the composition of a complete FTT with a restriction.

A *higher-order recursion scheme with states* (HORSS) is a triple $\mathcal{H} = (Q, (q_{init}, A_{init}), \mathcal{R})$, where Q is a finite set of control states, (q_{init}, A_{init}) is the *initial process* with q_{init} the *initial control state* and A_{init} the *initial nonterminal* that is of sort o, and $\mathcal{R}$ is a finite set of rules of the form

$$\text{(I)}\quad p, A^{\alpha_1\to\cdots\to\alpha_k\to o}\,x_1^{\alpha_1}\,\cdots\,x_k^{\alpha_k} \to q, K^o$$

$$\text{(II)}\quad p, a^r\,x_1^o\,\cdots\,x_r^o \to a^r\,(p_1,x_1)\,\cdots\,(p_r,x_r)$$

where the term K uses only variables from the set $\{x_1^{\alpha_1},\ldots,x_k^{\alpha_k}\}$. Rules of type (I) are as in standard HORS except that they are guarded by control states. Rules of type (II) correspond to a finite top-down tree automaton reading the tree produced by the HORS. The order of $\mathcal{S}$ is defined as the highest order of a nonterminal for which there is a rule in $\mathcal{S}$. Let us now describe the dynamics of HORSSes. A *process* is a pair (p, M) where M is a closed term of sort o and p is a state in Q. A *process tree* is a tree built of symbols and processes, where the latter are seen as symbols of rank 0. A HORSS $\mathcal{H}$ defines a reduction relation $\to_{\mathcal{H}}$ on process trees:

$$\frac{(p, A\,x_1\,\ldots\,x_k \to q, K) \in \mathcal{R}(\mathcal{H})}{(p, A\,M_1\,\ldots\,M_k) \to_{\mathcal{H}} (q, K[M_1/x_1,\ldots,M_k/x_k])}$$

$$\frac{(p, a^r\,x_1\,\cdots\,x_r \to a\,(p_1,x_1)\,\cdots\,(p_r,x_r)) \in \mathcal{R}(\mathcal{H})}{(p, a^r\,M_1\cdots M_r) \to_{\mathcal{H}} a\,(p_1,M_1)\cdots(p_r,M_r)}$$

$$\frac{K_l \to_{\mathcal{H}} K_l' \text{ for some } l \in \{1,\ldots,r\} \qquad K_i = K_i' \text{ for all } i \neq l}{a^r\,K_1\,\ldots\,K_r \to_{\mathcal{H}} a^r\,K_1'\,\ldots\,K_r'}$$

We are interested in finite trees generated by HORSSes. A process tree T is a *tree* if it does not contain any process. A HORSS $\mathcal{H}$ *generates* a tree T from a process (p, M) if $(p, M) \to_{\mathcal{H}}^* T$. The language $\mathcal{L}(\mathcal{H})$ is the set of trees generated by the initial process (q_{init}, A_{init}).

A HORS can be seen as a special case of a HORSS where Q has only one state $\hat{p}$ with the trivial rule $\hat{p}, a\,x_1\cdots x_k \to a\,(\hat{p},x_1)\cdots(\hat{p},x_k)$. It is well known that this extension does not increase expressive power of HORS, in the sense that given a HORSS $\mathcal{H}$ it is possible to construct a (standard) HORS $\mathcal{S}$ of the same order as $\mathcal{H}$ (but where the arity of nonterminals is increased) such that $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{S})$ [17]. However, while combining a HORS with an FTT it is convenient to create a HORSS, as its states can be used to simulate states of the FTT.

On the other hand, it is also useful to have the input HORS in a special normalized form, defined next. We say that a HORS is *normalized* if every its rule is of the form

$$A\,x_1\,\ldots\,x_p \to h\,(B_1\,x_1\,\ldots\,x_p)\,\ldots\,(B_r\,x_1\,\ldots\,x_p)\,,$$

where $r \geq 0$, h is either one of the x_i's, a nonterminal, or a symbol, and the B_j's are nonterminals. The arity p may be different in each rule. We will not detail the rather standard procedure of transforming any HORS into a normalized HORS without increasing the order. It amounts to splitting every rule into multiple rules, using fresh nonterminals in the cut points.

Lemma A.1. *HORSes are affectively closed under complete linear tree transductions.*

Proof. Let $\mathcal{S}$ be a HORS and let $\mathcal{A}$ be a linear FTT. We construct a HORSS $\mathcal{H}$ s.t. $\mathcal{L}(\mathcal{H}) = \mathcal{T}(\mathcal{A})(\mathcal{L}(\mathcal{S}))$. The set of control states of $\mathcal{H}$ is taken to be the set of control states of the FTT $\mathcal{A}$. As noted above, we can assume w.l.o.g. that $\mathcal{S}$ is normalized.

First, if $\mathcal{S}$ contains a rule $A\,\vec{x} \to h\,M_1\cdots M_r$ with h not a symbol, then $\mathcal{H}$ contains the rule $p, A\,\vec{x} \to p, h\,M_1\cdots M_r$ for every control state p.

Next, for every such rule with h being a symbol a^r, and for every transition of $\mathcal{A}$ having a^r on the left side, we take to $\mathcal{H}$ one rule illustrated by means of a representative example: if $\mathcal{A}$ contains a transition

$$a^2\,(p_1,x_1)\,(p_2,x_2) \longrightarrow p, b^2\,(c^1\,x_1)\,x_2$$

and $\mathcal{S}$ contains a rule $A\,\vec{y} \to a^2\,(B_1\,\vec{y})\,(B_2\,\vec{y})$, then $\mathcal{H}$ contains the rule

$$p, A\,\vec{y} \to b^2\,(c^1\,(p_1, B_1\,\vec{y}))\,(p_2, B_2\,\vec{y})$$

Technically speaking, this is not a HORSS rule, but it can be turned into one type (I) rule and several type (II) rules by adding new states.

Finally, we also add rules corresponding to ε-transitions of $\mathcal{A}$, what is again defined by an example: if $\mathcal{A}$ contains a transition

$$p, x_1 \longrightarrow q, a^1\,x_1$$

then, for every nonterminal A of $\mathcal{S}$, $\mathcal{H}$ contains the rule

$$q, A\,\vec{y} \to a^1\,(p, A\,\vec{y})$$

The two inclusions needed to show that $\mathcal{L}(\mathcal{H}) = \mathcal{T}(\mathcal{A})(\mathcal{L}(\mathcal{S}))$ can be proved straightforwardly by induction on the length of derivations. □

The difficulty in proving closure under possibly non-complete FTTs is that when combining a (non-complete) FTT transition of the form e.g. $a^2\,(p,x_1)\,(p,x_2) \longrightarrow p, b^1\,x_1$ with a HORS rule of the form e.g. $A\,\vec{y} \to a^2\,(B_1\,\vec{y})\,(B_2\,\vec{y})$, we cannot simply discard the subterm $B_2\,\vec{y}$, but we have to make sure that it generates at least one tree on which the FTT has some run. While concentrating on closure only under restrictions, one think becomes easier: a restriction has a run almost on every tree. There is, however, one exception: a restriction $\mathcal{A}$ does not have a run on a tree that uses a symbol for which $\mathcal{A}$ has no transition. We deal with this in Lemma A.2, below. However, knowing that on every tree there is a run of $\mathcal{A}$ is not enough; we also need to know that $B_2\,\vec{y}$ generates at least one tree. This problem is resolved by Lemma A.3.

Lemma A.2. *For every set of (ranked) symbols Θ and every HORS $\mathcal{S}$ we can build a HORS $\mathcal{S}'$ of the same order, such that $\mathcal{L}(\mathcal{S}')$ contains those trees from $\mathcal{L}(\mathcal{S})$ which use only symbols from Θ.*

Proof. We start by assuming w.l.o.g. that $\mathcal{S}$ is normalized. Then, we simply remove from $\mathcal{S}$ all rules that use symbols not in Θ. Then surely trees in $\mathcal{L}(\mathcal{S}')$ use only symbols from Θ. On the other hand, since $\mathcal{S}$ was normalized, every removed rule was of the form $A\,\vec{y} \to a^r\,(B_1\,\vec{y})\,\ldots\,(B_r\,\vec{y})$ (with $a^r \notin \Theta$), so whenever such a rule was used, an a^r-labeled node was created. In consequence, removing these rules has no influence on generating trees that use only symbols from Θ. □

A HORS $\mathcal{S} = (A_{init}, \mathcal{R})$ is *productive* if, whenever we can reduce A_{init} to a term M (which may contain nonterminals), then M can be reduced to some finite tree. By using the reflection operation [6], we can easily turn a HORS into a productive one.

Lemma A.3. *For every HORS $\mathcal{S}$ we can build a productive HORS $\mathcal{S}'$ of the same order generating the same trees.*

Proof. First, we construct a deterministic scheme $\mathcal{T}$ from the non-deterministic scheme $\mathcal{S}$. To $\mathcal{T}$ we will be then able to apply a reflection transformation. We use a letter $+$ to eliminate non-determinism. For every nonterminal A of $\mathcal{S}$ we collect all its rules: $A\,x_1\,\ldots\,x_p \to K_1, \ldots, A\,x_1\,\ldots\,x_p \to K_m$, and add to $\mathcal{T}$ the single rule:

$$A\,x_1\,\ldots\,x_p \to +^2\,K_1\,(+^2\,K_2\,(\ldots\,(+^2\,K_{m-1}\,K_m)\ldots))\,.$$

The (possibly infinite) tree generated by $\mathcal{T}$ represents the language of trees generated from $\mathcal{S}$ since the non-deterministic choices that can be made in $\mathcal{S}$ are represented by nodes labeled by $+$ in the tree generated by $\mathcal{T}$. In this latter tree, we can find every tree generated by $\mathcal{S}$ using a finite number of rewriting steps consisting of replacing a subtree rooted in $+$ by one of its children.

We now take the monotone applicative structure (see [22, 25]) $\mathcal{M} = (\mathcal{M}_\alpha)_{\alpha\in\mathrm{Sorts}}$ where $\mathcal{M}_o$ is the two element lattice, with maximal element $\top$ and minimal element $\bot$. Intuitively, $\top$ means nonempty language and $\bot$ means empty language. We interpret $+^2$ as the join (max) of its arguments, and every other symbol a^r as the meet (min) of its arguments; in particular symbols of rank 0 are interpreted as $\top$. This allows us to define the semantics $[\![M, \chi, \nu]\!]$ of a term given a valuation χ for nonterminals and ν for variables (these valuations assign to a variable/nonterminal a value in $\mathcal{M}$ of an appropriate sort). The definition of $[\![M, \chi, \nu]\!]$ is standard, in particular $[\![K\,L, \chi, \nu]\!] = [\![K, \chi, \nu]\!]([\![L, \chi, \nu]\!])$.

The meaning of nonterminals in $\mathcal{T}$ is given by the least fixpoint computation. For a valuation χ of the nonterminals of $\mathcal{T}$, we write $\mathcal{T}(\chi)$ for the valuation χ' such that $\chi'(A) = \lambda g_1.\cdots.\lambda g_p.[\![K, \chi, [g_1/x_1, \ldots, g_p/x_p]]\!]$ where $A\,x_1\,\ldots\,x_p \to K$ is the rule for A in $\mathcal{T}$. Then the meaning of nonterminals is given by the valuation that is the least fixpoint of this operator: $\chi_\mathcal{T} = \bigwedge\{\chi : \mathcal{T}(\chi) \subseteq \chi\}$. Having $\chi_\mathcal{T}$ we can define the semantics of a term M in a valuation ν of its free variables as $[\![M, \nu]\!] = [\![M, \chi_\mathcal{T}, \nu]\!]$.

Least fixed point models of schemes induce an interpretation on infinite trees by finite approximations. An infinite tree has value $\top$ iff it represents a non-empty language [22]. The important point is that the semantics of a term and that of the infinite tree generated from the term coincide.

We can now apply to $\mathcal{T}$ the reflection operation [6] with respect to the above interpretation $\mathcal{M}$. The result is a scheme $\mathcal{T}'$ that generates the same tree as $\mathcal{T}$ but where every node is additionally marked by a tuple $(a_1, \ldots, a_r, b)$ where $a_1, \ldots, a_r$ is the semantics of the arguments of that node (i.e., subtrees rooted at its children) and b is the semantics of the subtree rooted at that node. What is important here is that $\mathcal{T}'$ has the same order as $\mathcal{T}$ which is the same as that of $\mathcal{S}$. The additional labels allow us to remove unproductive parts of the tree generated by $\mathcal{T}'$. For this we introduce two more nonterminals Π_1 and Π_2 of sort $o \to o \to o$. We then add the rules $\Pi_1\,x_1\,x_2 \to x_1, \Pi_2\,x_1\,x_2 \to x_2$. Now we replace every occurrence of $+^2$ labeled by $(\top, \bot, \top)$ by Π_1, and every occurrence of $+^2$ labeled by $(\bot, \top, \top)$ by Π_2. After these transformations we obtain a scheme $\mathcal{T}''$ generating a tree which contains exactly those nodes of $\mathcal{T}'$ that are labeled with $(\top, \ldots, \top, \top)$.

We convert $\mathcal{T}''$ into a HORS $\mathcal{S}'$ whose language is the same as that of $\mathcal{S}$. For this we replace every remaining occurrence of $+^2$ (thus labeled by $(\top, \top, \top)$) by a nonterminal C of sort $o \to o \to o$, and we add two rewrite rules $C\,x\,y \to x$ and $C\,x\,y \to y$. We also remove the additional labels from symbols. By construction, $\mathcal{S}'$ is productive and $\mathcal{L}(\mathcal{S}') \subseteq \mathcal{L}(\mathcal{S})$. Moreover, since we only eliminated non-productive nonterminals, $\mathcal{L}(\mathcal{S}') = \mathcal{L}(\mathcal{S})$. □

Lemma A.4. *Let $\mathcal{S}$ be a productive HORS, and $\mathcal{A}$ a restriction such that for every symbol a^r appearing in any tree generated by $\mathcal{S}$ there is a transition of $\mathcal{A}$ having a^r on the left side. Then we can build a HORS $\mathcal{S}'$ whose language is $\mathcal{T}(\mathcal{A})(\mathcal{L}(\mathcal{S}))$.*

Proof. First, w.l.o.g. we assume that $\mathcal{S}$ is normalized (notice that while converting a productive HORS to a normalized one, it remains productive). Every rule $S\,\vec{y} \to h\,(B_1\,\vec{y})\,\ldots\,(B_r\,\vec{y})$ of S in which h is not a symbol is also taken to $\mathcal{S}'$. If $h = a^r$ is a symbol, we consider every transition of $\mathcal{A}$ having a^r on the left side. Since $\mathcal{A}$ is a restriction, this transition is of the form

$$a^r\,(p, x_1)\,\ldots\,(p, x_r) \longrightarrow p, b^n\,x_{i_1}\cdots x_{i_n}\,,$$

where $1 \le i_1 < \cdots < i_n \le r$. Then, to $\mathcal{S}'$ we take the rule

$$A\,\vec{y} \to b^n\,(B_{i_1}\,\vec{y})\cdots(B_{i_n}\,\vec{y})\,.$$

In general, $\mathcal{T}(\mathcal{A})(\mathcal{L}(\mathcal{S})) \subseteq \mathcal{L}(\mathcal{S}')$. Since $\mathcal{S}$ is productive, the subterms $B_i\,\vec{y}$ obtained by rewriting the initial nonterminal A_{init} produce at least one tree, and since for every symbol in this tree there is a transition of $\mathcal{A}$ having this symbol on the left side, $\mathcal{A}$ has some run on this tree. Thus $\mathcal{T}(\mathcal{A})(\mathcal{L}(\mathcal{S})) = \mathcal{L}(\mathcal{S}')$. □

Theorem 2.1. *HORSes are effectively closed under linear tree transductions.*

Proof. A transduction $\mathcal{A}$ realized by an FTT is the composition of a complete one $\mathcal{B}$ and a restriction $\mathcal{C}$. We first apply Lemma A.1 to the complete transduction realized by $\mathcal{B}$. Then, using Lemma A.2 we remove from the generated language all trees that use symbols not appearing on the left side of any transition of $\mathcal{C}$. Next, we turn the resulting HORS into a productive one by Lemma A.3, and, finally, we apply Lemma A.4 to the resulting productive HORS and the restriction realized by $\mathcal{C}$. We end up with a HORS producing the image of $\mathcal{A}$ applied to the original HORS, and being of the same order. □

B. Proof of Lemma 5.14

We recall the the statement of the lemma.

Lemma 5.14. *Let D' be a maximal derivation for $\vdash L' : (S, \mathbf{r})$, and let L be a term that does not contain the initial nonterminal of $\mathcal{S}$ and such that $L \to_\mathcal{S} L'$. Then there exists a maximal derivation D for $\vdash L : (S, \mathbf{r})$ and a term P that is $merge$-equivalent to $tr_{cum}(D')$ and such that $merge(tr_{cum}(D)) \to_{\mathcal{S}'} P$.*

Proof. We proceed by induction on the structure of L.

Suppose first that $L = a^r\,M_1\,\ldots\,M_r$ (where surely $r \ge 1$). Then $L' = a^r\,M_1'\,\ldots\,M_r'$, where $M_l \to_\mathcal{S} M_l'$ for some $l \in \{1, \ldots, r\}$, and $M_i = M_i'$ for all $i \ne l$. The derivation D' contains a node labeled by $\vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})$, and for each $i \in \{1, \ldots, r\}$ a subtree D_i' deriving $\vdash M_i' : (S_i, \mathbf{r})$ (they are merged together by using the application rule r times), where $S_1, \ldots, S_r$ are disjoint and their union is S. We apply the induction assumption to M_l, obtaining a derivation D_l for $\vdash M_l : (S_l, \mathbf{r})$ and a term P_l $merge$-equivalent to $tr_{cum}(D_l')$ and such that $merge(tr_{cum}(D_l)) \to_{\mathcal{S}'} P_l$. We can write $P_l = merge(list_l')$ (where the length of $list_l'$ and $tr_{cum}(D_l)$ is the same). We take $D_i = D_i'$ for $i \ne l$, and out of the single-node derivation for $\vdash a^r : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})$ and of derivations D_i for $i \in \{1, \ldots, r\}$ we compose a derivation D, using the application rule r times. We see that $tr_{cum}(D) = (a^0; tr_{cum}(D_1); \ldots; tr_{cum}(D_r))$, and $tr_{cum}(D') = (a^0; tr_{cum}(D_1'); \ldots; tr_{cum}(D_r'))$. Moreover, taking $list_i' = tr_{cum}(D_i)$ for $i \ne l$ we get $merge(tr_{cum}(D)) \to_{\mathcal{S}'} merge(a^0; list_1'; \ldots; list_r')$, where $merge(a^0; list_1'; \ldots; list_r')$ is $merge$-equivalent to $tr_{cum}(D')$. It remains to observe that D is

maximal. Indeed, the nodes inside some D_i have all required children since D_i are maximal, and the new internal nodes created in D describe applications with an argument M_i of sort o, and it is impossible to derive $\vdash M_i : (\emptyset, \sigma)$ for any σ (since $\mathcal{LT}^o$ does not contain pairs with empty set on the first coordinate).

The remaining possibility is that $L = A\, N_1^{\alpha_1} \ldots N_k^{\alpha_k}$. Let $A\, x_1 \ldots x_k \to K$ be the rule of $\mathcal{S}$ used in the reduction $L \to_{\mathcal{S}} L'$, that is such that $L' = K[N_1/x_1, \ldots, N_k/x_k]$. Take $D_{0,K} = D'$. For $i \in \{1, \ldots, k\}$, consecutively, we apply Lemma 5.13 to $D_{i-1,K}$ and N_i, creating sets $\Lambda_i \subseteq \mathcal{LT}^{\alpha_i}$ and maximal derivations $D_{i,K}$ and $D_{i,\lambda}$ for $\lambda \in \Lambda_i$. Let $D_K = D_{k,K}$; it derives $\Gamma \vdash K : (S, \mathbf{r})$, where $\Gamma = \{x_i : \lambda \mid i \in \{1, \ldots, k\}, \lambda \in \Lambda_i\}$. By point 2 of Lemma 5.13 we know that for every $\lambda \in \Lambda_i$ with a nonempty set on the first coordinate, in D_K there is a node labeled by $\Gamma \vdash x_i : \lambda$. On the one hand, since our type systems requires that subsets of Σ_0 coming from different children are disjoint, we can be sure that the sets on the first coordinate of labeled types in $\Lambda_1, \ldots, \Lambda_k$ are disjoint. It follows that $\tau_A = \Lambda_1 \to \cdots \to \Lambda_k \to \mathbf{r}$ is a type. On the other hand, nodes labeled by $\Gamma \vdash x_i : \lambda$ give the only possibility for introducing elements of S to our derivation D_K (by assumption in K we do not have nullary symbols, since A is not the initial nonterminal), which means that the union of the sets on the first coordinate of labeled types in $\Lambda_1, \ldots, \Lambda_k$ is S. Since $(S, \mathbf{r}) \in \mathcal{LT}^o$, we have $S \neq \emptyset$, and thus $k \geq 1$, which means that $(\emptyset, \tau_A)$ is a labeled type.

In order to obtain the required derivation D for $\vdash L : (S, \mathbf{r})$, we start with the single-node derivation for $\vdash A : (\emptyset, \tau_A)$, and using the application rule k times we attach derivations $D_{i,\lambda}$ for each $i \in \{1, \ldots, k\}$ and $\lambda \in \Lambda_i$. This derivation is maximal, since $D_{i,\lambda}$ were maximal, and by point 1 of Lemma 5.13 the newly created internal nodes have all required children (whenever it is possible to derive a type judgment $\vdash N_i : (\emptyset, \sigma)$, we are deriving it in D).

Recall that $tr_{cum}(D)$ is a concatenation of $tr(D)$ and of $tr_{cum}(D_{i,\lambda})$ for every $i \in \{1, \ldots, k\}$ and $\lambda \in \Lambda_i$. For $i \in \{1, \ldots, k\}$ let η_i be the substitution that maps $x_i\restriction_\lambda$ to $tr(D_{i,\lambda})$ for every $\lambda \in \Lambda_i$. In $\mathcal{S}'$ we have the rule $A_\tau\, \overline{\mathbf{x_1}} \ldots \overline{\mathbf{x_k}} \to merge(tr_{cum}(D_K))$, where $\overline{\mathbf{x_i}}$ lists variables $x_i\restriction_\lambda$ for $\lambda \in \Lambda_i$ if $\alpha_i \neq o$, and is empty if $\alpha_i = o$ (for $i \in \{1, \ldots, k\}$). Notice that this rule applied to $tr(D)$ gives $merge(tr_{cum}(D_K))[\eta_1, \ldots, \eta_k]$ (substitutions η_i for i such that $\alpha_i = o$ can be skipped, since anyway variables $x_i\restriction_{\lambda_{i,j}}$ for such i do not appear in $tr_{cum}(D_K)$). As P we take $merge(\cdot)$ of the concatenation of this term and of all $tr_{cum}(D_{i,\lambda})$; as we have said $merge(tr_{cum}(D)) \to_{\mathcal{S}'} P$. From point 3 of Lemma 5.13 it follows that $tr_{cum}(D')$ is $merge$-equivalent to P, what finishes the proof. □