Higher-Order Nonemptiness Step by Step

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

Abstract

We show a new simple algorithm that checks whether a given higher-order grammar generates a nonempty language of trees. The algorithm amounts to a procedure that transforms a grammar of order n to a grammar of order $n-1$, preserving nonemptiness, and increasing the size only exponentially. After repeating the procedure n times, we obtain a grammar of order 0, whose nonemptiness can be easily checked. Since the size grows exponentially at each step, the overall complexity is n-EXPTIME, which is known to be optimal. More precisely, the transformation (and hence the whole algorithm) is linear in the size of the grammar, assuming that the arity of employed nonterminals is bounded by a constant. The same algorithm allows to check whether an infinite tree generated by a higher-order recursion scheme is accepted by an alternating safety (or reachability) automaton, because this question can be reduced to the nonemptiness problem by taking a product of the recursion scheme with the automaton.

A proof of correctness of the algorithm is formalised in the proof assistant Coq. Our transformation is motivated by a similar transformation of Asada and Kobayashi (2020) changing a word grammar of order n to a tree grammar of order $n-1$. The step-by-step approach can be opposed to previous algorithms solving the nonemptiness problem "in one step", being compulsorily more complicated.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases Higher-order grammars, Nonemptiness, Model-checking, Transformation, Order reduction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.53

Related Version A full version of the paper is available at `https://arxiv.org/abs/2009.08174`.

Supplementary Material Coq formalisation: `https://github.com/pparys/ho-transform-sbs`

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

1 Introduction

Higher-order grammars, also known as higher-order OI grammars [8, 16], generalize context-free grammars: nonterminals of higher-order grammars are allowed to take arguments. Such grammars have been studied actively in recent years, in the context of automated verification of higher-order programs. In this paper we concentrate on a very basic problem of language nonemptiness: is the language generated by a given higher-order grammar nonempty. This problem, being easy for most devices, is not so easy for higher-order grammars. Indeed, it is n-EXPTIME-complete for grammars of order n [15].

We give a new simple algorithm solving the language nonemptiness problem. The algorithm amounts to a procedure that transforms a grammar of order n to a grammar of order $n-1$, preserving nonemptiness, and increasing the size only exponentially. After repeating the procedure n times, we obtain a grammar of order 0, whose nonemptiness can be easily checked. Since the size grows exponentially at each step, we reach the optimal overall complexity of n-EXPTIME. In a more detailed view, the complexity looks even better: the size growth is exponential only in the arity of types appearing in the grammar; if the maximal arity is bounded by a constant, the transformation (and hence the whole algorithm) is linear in the size of the grammar.

While a higher-order grammar is a generator of a language of (finite) trees, virtually the same object can be seen as a generator of a single infinite tree (encompassing the whole language). In this context, the grammars are called higher-order recursion schemes. The nonemptiness problem for grammars is easily equivalent to the question whether the tree generated by a given recursion scheme is accepted by a given alternating safety (or reachability) automaton; for the right-to-left reduction, it is enough to product the recursion scheme with the automaton. Thus, our algorithm solves also the latter problem, called a model-checking problem. This problem is decidable and n-EXPTIME-complete not only for safety or reachability automata, but actually for all parity automata, with multiple proofs using game semantics [17], collapsible pushdown automata [10], intersection types [14], or Krivine machines [20], and with several extensions [5, 3, 6, 21, 18]. The problem for safety automata was tackled in particular by Aehlig [1] and by Kobayashi [12]. To those algorithms we add another one. The main difference between our algorithm and all the others is that we solve the problem step by step, repeatedly reducing the order by one, while most previous algorithms work "in one step", being compulsorily more complicated. The only proofs that have been reducing the order by one, were proofs using collapsible pushdown automata [10, 3, 6], being very technical (and contained only in unpublished appendices). A reduction of order was also possible for a subclass of recursion schemes, called *safe* recursion schemes [11], but it was not known how to extend it to all recursion schemes.

Comparing the two variants of the model-checking problem for higher-order recursion schemes – involving safety and reachability automata, and involving all parity automata – we have to mention two things. First, while most theoretical results can handle all parity automata, actual tools solving this problem in practice mostly deal only with safety and reachability automata (called also trivial and co-trivial automata) [13, 4, 22, 19]. Second, there exists a polynomial-time (although nontrivial) reduction from the variant involving parity automata to the variant involving safety automata [9].

Our transformation is directly motivated by a recent paper of Asada and Kobayashi [2]. They show how to transform a grammar of order n generating a language of words to a grammar of order $n-1$ generating a language of trees, so that words of the original language are written in leaves of trees of the new language. Unexpectedly, this transformation increases the size of the grammar only polynomially. Our transformation is quite similar, but we start from a grammar generating a language of trees, not words. In effect, on the one hand, we do not say anything specific about the language after the transformation (except that nonemptiness is preserved), and on the other hand, the size growth is exponential, not polynomial.

2 Preliminaries

For a number $k \in \mathbb{N}$ we write $[k]$ for $\{1, \dots, k\}$.

The set of *(simple) types* is constructed from a unique ground type o using a binary operation $\to$; namely o is a type, and if α and β are types, so is $\alpha \to \beta$. By convention, $\to$ associates to the right, that is, $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$. We often abbreviate $\underbrace{\alpha \to \cdots \to \alpha}_{\ell} \to \beta$ as $\alpha^\ell \to \beta$. The *order* of a type α, denoted $\mathsf{ord}(\alpha)$, is defined by induction: $\mathsf{ord}(\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}) = \max(\{0\} \cup \{\mathsf{ord}(\alpha_i) + 1 \mid i \in [k]\})$; for example $\mathsf{ord}(\mathsf{o}) = 0$, $\mathsf{ord}(\mathsf{o} \to \mathsf{o} \to \mathsf{o}) = 1$, and $\mathsf{ord}((\mathsf{o} \to \mathsf{o}) \to \mathsf{o}) = 2$.

Having a finite set of typed nonterminals $\mathcal{X}$, and a finite set of typed variables $\mathcal{Y}$, *terms* over $(\mathcal{X}, \mathcal{Y})$ are defined by induction:

- every nonterminal $X \in \mathcal{X}$ of type α is a term of type α;

- every variable $y \in \mathcal{Y}$ of type α is a term of type α;
- if $K_1, \ldots, K_k$ are terms of type o, then $\bullet\langle K_1, \ldots, K_k\rangle$ and $\oplus\langle K_1, \ldots, K_k\rangle$ are terms of type o;
- if K is a term of type $\alpha \to \beta$, and L is a term of type α, then $K\,L$ is a term of type β.

The type of a term K is denoted $\mathsf{tp}(K)$. The order of a term K, written $\mathsf{ord}(K)$, is defined as the order of its type. We write Ω for $\oplus\langle\rangle$, and $\bullet$ for $\bullet\langle\rangle$.

The construction $\oplus\langle K_1, \ldots, K_k\rangle$ is an alternative; such a term reduces to one of the terms $K_1, \ldots, K_k$. This construction is used to introduce nondeterminism to grammars (defined below). In the special case of $k = 0$ (when we write Ω) no reduction is possible; thus Ω denotes divergence.

The construction $\bullet\langle K_1, \ldots, K_k\rangle$ can be seen as a generator of a tree node with k children; subtrees starting in these children are described by the terms $K_1, \ldots, K_k$. In a usual presentation, nodes are labeled by letters from some finite alphabet. In this paper, however, we do not care about the exact letters contained in generated trees, only about language nonemptiness, hence we do not write these letters at all (in other words, we use a single-letter alphabet, where $\bullet$ is the only letter). Actually, in the sequel we even do not consider trees; we rather say that $\bullet\langle K_1, \ldots, K_k\rangle$ is convergent if all $K_1, \ldots, K_k$ are convergent (which can be rephrased as: the language generated from $\bullet\langle K_1, \ldots, K_k\rangle$ is nonempty if the languages generated from all $K_1, \ldots, K_k$ are nonempty).

A *(higher-order) grammar* is a tuple $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$, where $\mathcal{X}$ a finite set of typed nonterminals, $X_0 \in \mathcal{X}$ is a starting nonterminal of type o, and $\mathcal{R}$ a function assigning to every nonterminal $X \in \mathcal{X}$ a rule of the form $X\,y_1\,\ldots\,y_k \to R$, where $\mathsf{tp}(X) = (\mathsf{tp}(y_1) \to \cdots \to \mathsf{tp}(y_k) \to \mathsf{o})$, and R is a term of type o over $(\mathcal{X}, \{y_1, \ldots, y_k\})$. The order of a grammar is defined as the maximum of orders of its nonterminals.

Having a grammar $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$, for every set of variables $\mathcal{Y}$ we define a *reduction relation* $\longrightarrow_{\mathcal{G}}$ between terms over $(\mathcal{X}, \mathcal{Y})$ and sets of such terms, as the least relation such that

(1) $X\,K_1\,\ldots\,K_k \longrightarrow_{\mathcal{G}} \{R[K_1/y_1, \ldots, K_k/y_k]\}$ if the rule for X is $X\,y_1\,\ldots\,y_k \to R$, where $R[K_1/y_1, \ldots, K_k/y_k]$ denotes the term obtained from R by substituting K_i for y_i for all $i \in [k]$,

(2) $\bullet\langle K_1, \ldots, K_k\rangle \longrightarrow_{\mathcal{G}} \{K_1, \ldots, K_k\}$, and

(3) $\oplus\langle K_1, \ldots, K_k\rangle \longrightarrow_{\mathcal{G}} \{K_i\}$ for every $i \in [k]$.

We say that a term M is *$\mathcal{G}$-convergent* if $M \longrightarrow_{\mathcal{G}} \mathcal{N}$ for some set $\mathcal{N}$ of $\mathcal{G}$-convergent terms. This is an inductive definition; in particular, the base case is when $M \longrightarrow_{\mathcal{G}} \emptyset$. In other words, M is *$\mathcal{G}$-convergent* if there is a finite tree labeled by terms where for each node, the node and its children satisfy one of (1)-(3). Moreover, the grammar $\mathcal{G}$ is *convergent* if its starting nonterminal X_0 is $\mathcal{G}$-convergent.

3 Transformation

In this section we present a transformation, called *order-reducing transformation*, resulting in the main theorem of this paper:

▶ **Theorem 3.1.** *For any $n \geq 1$, there exists a transformation from order-n grammars to order-$(n-1)$ grammars, and a polynomial p_n such that, for any order-n grammar $\mathcal{G}$, the resulting grammar $\mathcal{G}^\dagger$ is convergent if and only if $\mathcal{G}$ is convergent, and $|\mathcal{G}^\dagger| \leq 2^{p_n(|\mathcal{G}|)}$.*

Intuitions. Let us first present intuitions behind our transformation. While reducing the order, we have to replace, in particular, order-1 functions by order-0 terms. Consider for example a term $K\,L$ of type o, where K has type $\mathsf{o} \to \mathsf{o}$. Notice that L generates trees that are inserted somewhere in contexts generated by K. Thus, when is $K\,L$ convergent? There are two possibilities. First, maybe K is convergent without using its argument at all. Second, maybe K can be convergent but only using its argument, and then L also has to be convergent. Notice that in the first case $K\,\Omega$ is convergent (i.e., K is convergent even if the argument is not convergent), and in the second case $K\,\bullet$ is convergent (i.e., K is convergent if its argument is convergent). In the transformation, we transform K into two order-0 terms, K_0 and K_1 corresponding to $K\,\Omega$ and $K\,\bullet$, and then we replace $K\,L$ by $\oplus\langle K_0, \bullet\langle K_1, L\rangle\rangle$.

As a full example, consider an order-1 grammar with the following rules:

$$\mathsf{X} \to \mathsf{Y}\,\mathsf{Z}, \qquad \mathsf{Y}\,\mathsf{x} \to \oplus\langle \bullet, \mathsf{x}\rangle, \qquad \mathsf{Z} \to \bullet.$$

It will be transformed to the order-0 grammar with the following rules:

$$\mathsf{X} \to \oplus\langle \mathsf{Y}_0, \bullet\langle \mathsf{Y}_1, \mathsf{Z}\rangle\rangle, \qquad \mathsf{Y}_0 \to \oplus\langle \bullet, \Omega\rangle, \qquad \mathsf{Y}_1 \to \oplus\langle \bullet, \bullet\rangle, \qquad \mathsf{Z} \to \bullet.$$

Notice that the original grammar is convergent "for two reasons": the $\oplus$ node in the rule for Y may reduce either to the first possibility (i.e., to $\bullet$), or to the second possibility (i.e., to x), in which case convergence follows from convergence of the argument Z. This is reflected by the two possibilities available for the $\oplus$ node in the new rule for X: we either choose the first possibility and we depend only on convergence of Y_0, or we choose the the second possibility and we depend on convergence of both Y_1 and Z. Notice that after replacing the (old and new) rule for Z by $\mathsf{Z} \to \Omega$, the modified grammars remain convergent thanks to the first possibility above. Likewise, after replacing the original rule for Y by $\mathsf{Y}\,\mathsf{x} \to \mathsf{x}$, the new rules will be $\mathsf{Y}_0 \to \Omega$ and $\mathsf{Y}_1 \to \bullet$, and the modified grammars remain convergent thanks to the second possibility above. However, after applying both these replacements simultaneously, the grammars stop to be convergent.

If our term K takes multiple order-0 arguments, say we have $K\,L_1\,\ldots\,L_k$, while transforming K we need 2^k variants of the term: each of the arguments may be either used (replaced by $\bullet$) or not used (replaced by Ω). This is why we have the exponential blow-up. Let us compare this quickly with the transformation of Asada and Kobayashi [2], which worked for grammars generating words (i.e., trees where every node has at most one child). In their case, at most one of the arguments L_i could be used, so they needed only $k+1$ variants of K; this is why their transformation was polynomial.

For higher-order grammars we apply the same idea: functions of order 1 are replaced by terms of order 0, and then the order of any higher-order function drops down by one. For example, consider a grammar with the following rules:

$$\mathsf{X} \to \mathsf{T}\,\mathsf{Y}, \qquad \mathsf{T}\,\mathsf{y} \to \mathsf{y}\,(\mathsf{y}\,\bullet), \qquad \mathsf{Y}\,\mathsf{x} \to \oplus\langle \bullet, \mathsf{x}\rangle.$$

The nonterminal Y is again of type $\mathsf{o} \to \mathsf{o}$, hence it is replaced by two nonterminals $\mathsf{Y}_0, \mathsf{Y}_1$ of type o, describing the situation when the parameter x is either not used or used. Likewise, the corresponding parameter y of T is replaced by two parameters $\mathsf{y}_0, \mathsf{y}_1$. The resulting grammar will have the following rules:

$$\mathsf{X} \to \mathsf{T}\,\mathsf{Y}_0\,\mathsf{Y}_1, \quad \mathsf{T}\,\mathsf{y}_0\,\mathsf{y}_1 \to \oplus\langle \mathsf{y}_0, \bullet\langle \mathsf{y}_1, \oplus\langle \mathsf{y}_0, \bullet\langle \mathsf{y}_1, \bullet\rangle\rangle\rangle\rangle, \quad \mathsf{Y}_0 \to \oplus\langle \bullet, \Omega\rangle, \quad \mathsf{Y}_1 \to \oplus\langle \bullet, \bullet\rangle.$$

Formal definition. We now formalize the above intuitions. Having a type, we are interested in cutting off its suffix being of order 1. Thus, we use the notation $\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o}$ for a type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}^\ell \to \mathsf{o}$ such that either $k = 0$ or $\alpha_k \neq \mathsf{o}$. Notice that every

type α can be uniquely represented in this form. We remark that some among the types $\alpha_1, \ldots, \alpha_{k-1}$ (but not α_k) may be o. For a type α we write $\mathsf{gar}(\alpha)$ ("ground arity") for the number ℓ for which we can write $\alpha = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})$; we also extend this to terms: $\mathsf{gar}(M) = \mathsf{gar}(\mathsf{tp}(M))$.

We transform terms of type α to terms of type $\alpha^\dagger$, which is defined by induction:

$$(\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})^\dagger = \left((\alpha_1^\dagger)^{2^{\mathsf{gar}(\alpha_1)}} \to \cdots \to (\alpha_k^\dagger)^{2^{\mathsf{gar}(\alpha_k)}} \to \mathsf{o}\right).$$

Thus, we remove all trailing order-0 arguments, and we multiplicate (and recursively transform) remaining arguments.

For a finite set S, we write 2^S for the set of functions $A\colon S \to \{0,1\}$. Moreover, we assume some fixed order on functions in 2^S, and we write $P\,(Q_A)_{A\in 2^S}$ for an application $P\,Q_{A_1}\,\ldots\,Q_{A_{2^{|S|}}}$, where $A_1, \ldots, A_{2^{|S|}}$ are all the function from 2^S listed in the fixed order. The only function in $2^\emptyset$ is denoted $\emptyset$.

Fix a grammar $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$. For every nonterminal X and for every function $A \in 2^{[\mathsf{gar}(X)]}$ we consider a nonterminal $X_A^\dagger$ of type $(\mathsf{tp}(X))^\dagger$. As the new set of nonterminals we take $\mathcal{X}^\dagger = \{X_A^\dagger \mid X \in \mathcal{X}, A \in 2^{[\mathsf{gar}(X)]}\}$. Likewise, for every variable y and for every function $A \in 2^{[\mathsf{gar}(y)]}$ we consider a variable $y_A^\dagger$ of type $(\mathsf{tp}(y))^\dagger$, and for a set of variables $\mathcal{Y}$ we denote $\mathcal{Y}^\dagger = \{y_A^\dagger \mid y \in \mathcal{Y}, A \in 2^{[\mathsf{gar}(y)]}\}$.

We now define a function tr transforming terms. Its value $\mathsf{tr}(A, Z, M)$ is defined when M is a term over some $(\mathcal{X}, \mathcal{Y})$, and $A \in 2^{[\mathsf{gar}(M)]}$, and $Z\colon \mathcal{Y} \rightharpoonup \{0,1\}$ is a partial function such that $\mathrm{dom}(Z)$ contains only variables of type o. The intention is that A specifies which among trailing order-0 arguments can be used, and Z specifies which order-0 variables (among those in $\mathrm{dom}(Z)$) can be used. The transformation is defined by induction on the structure of M, as follows:

(1) $\mathsf{tr}(A, Z, X) = X_A$ for $X \in \mathcal{X}$;

(2) $\mathsf{tr}(A, Z, y) = y_A$ for $y \in \mathcal{Y} \setminus \mathrm{dom}(Z)$;

(3) $\mathsf{tr}(A, Z, z) = \Omega$ if $Z(z) = 0$;

(4) $\mathsf{tr}(A, Z, z) = \bullet$ if $Z(z) = 1$;

(5) $\mathsf{tr}(\emptyset, Z, \bullet\langle K_1, \ldots, K_k\rangle) = \bullet\langle \mathsf{tr}(\emptyset, Z, K_1), \ldots, \mathsf{tr}(\emptyset, Z, K_k)\rangle$;

(6) $\mathsf{tr}(\emptyset, Z, \oplus\langle K_1, \ldots, K_k\rangle) = \oplus\langle \mathsf{tr}(\emptyset, Z, K_1), \ldots, \mathsf{tr}(\emptyset, Z, K_k)\rangle$;

(7) $\mathsf{tr}(A, Z, K\,L) = \oplus\langle \mathsf{tr}(A[\ell+1 \mapsto 0], Z, K), \bullet\langle \mathsf{tr}(A[\ell+1 \mapsto 1], Z, K), \mathsf{tr}(\emptyset, Z, L)\rangle\rangle$ if $\mathsf{tp}(K) = (\mathsf{o}^{\ell+1} \to \mathsf{o})$;

(8) $\mathsf{tr}(A, Z, K\,L) = (\mathsf{tr}(A, Z, K))\,(\mathsf{tr}(B, Z, L))_{B\in 2^{[\mathsf{gar}(L)]}}$ if $\mathsf{tp}(K) = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})$ with $k \geq 1$.

For every rule $X\,y_1\,\ldots\,y_k\,z_1\,\ldots\,z_\ell \to R$ in $\mathcal{R}$, where $\ell = \mathsf{gar}(X)$, and for every function $A \in 2^{[\ell]}$, to $\mathcal{R}^\dagger$ we take the rule

$$X_A^\dagger\,(y_{1,B}^\dagger)_{B\in 2^{[\mathsf{gar}(y_1)]}}\,\ldots\,(y_{k,B}^\dagger)_{B\in 2^{[\mathsf{gar}(y_k)]}} \to \mathsf{tr}(\emptyset, [z_i \mapsto A(\ell+1-i) \mid i \in [\ell]], R).$$

In the function A it is more convenient to count arguments from right to left (then we do not need to shift the domain in Case (7) above), but it is more natural to have variables $z_1, \ldots, z_\ell$ numbered from left to right; this is why in the rule for $X_A^\dagger$ we assign to z_i the value $A(\ell+1-i)$, not $A(i)$.

Finally, the resulting grammar $\mathcal{G}^\dagger$ is $(\mathcal{X}^\dagger, X_{0,\emptyset}^\dagger, \mathcal{R}^\dagger)$.

4 Complexity

In this section we analyze complexity of our transformation. First, we formally define the *size* of a grammar. The size of a term is defined by induction on its structure:

$$|X| = |y| = 1, \qquad |K\,L| = 1 + |K| + |L|,$$
$$|\bullet\langle K_1, \ldots, K_k\rangle| = |\oplus\langle K_1, \ldots, K_k\rangle| = 1 + |K_1| + \cdots + |K_k|.$$

Then $|\mathcal{G}|$, the size of $\mathcal{G}$ is defined as the sum of $|R| + k$ over all rules $X\,y_1\,\ldots\,y_k \to R$ of $\mathcal{G}$. In Asada and Kobayashi [2] such a size is called *Curry-style* size; it does not include sizes of types of employed variables.

We say that a type α is a *subtype* of a type β if either $\alpha = \beta$, or $\beta = (\beta_1 \to \beta_2)$ and α is a subtype of β_1 or of β_2. We write $A_{\mathcal{G}}$ for the largest arity of subtypes of types of nonterminals in a grammar $\mathcal{G}$. Notice that types of other objects appearing in $\mathcal{G}$, namely variables and subterms of right sides of rules, are subtypes of types of nonterminals, hence their arity is also bounded by $A_{\mathcal{G}}$. It is reasonable to consider large grammars, consisting of many rules, where simultaneously the maximal arity $A_{\mathcal{G}}$ is respectively small.

While the exponential bound mentioned in Theorem 3.1 is obtained by applying the order-reducing transformation to an arbitrary grammar, the complexity becomes slightly better if we first apply a preprocessing step. This is in particular necessary, if we want to obtain linear dependence in the size of $\mathcal{G}$ (and exponential only in the maximal arity $A_{\mathcal{G}}$). The preprocessing, making sure that the grammar is in a *simple form* (defined below) amounts to splitting large rules into multiple smaller rules. A similar preprocessing is present already in prior work [13, 2, 7], however our definition of a simple form is slightly more liberal, so that the order reduction applied to a grammar in a normal form gives again a grammar in a normal form.

An *application depth* of a term R is defined as the maximal number of applications on a single branch in R, where a compound application $K\,L_1\,\ldots\,L_k$ counts only once. More formally, we define by induction:

$$\mathsf{ad}(\bullet\langle K_1, \ldots, K_k\rangle) = \mathsf{ad}(\oplus\langle K_1, \ldots, K_k\rangle) = \max\{\mathsf{ad}(K_i) \mid i \in [k]\},$$
$$\mathsf{ad}(X\,K_1\,\ldots\,K_k) = \mathsf{ad}(y\,K_1\,\ldots\,K_k) = \max(\{0\} \cup \{\mathsf{ad}(K_i) + 1 \mid i \in [k]\}).$$

We say that a grammar $\mathcal{G}$ is in a *simple form* if the right side of each its rule has application depth at most 2.

Any grammar $\mathcal{G}$ can be converted to a grammar in a simple form, as follows. Consider a rule $X\,y_1\,\ldots\,y_k \to R$, and a subterm of R of the form $f\,K_1\,\ldots\,K_m$, where f is a nonterminal or a variable, but some K_i already has application depth 2. Then we replace the occurrence of K_i with $Y\,y_1\,\ldots\,y_k$ (being a term of application depth 1) for a fresh nonterminal Y, and we add the rule $Y\,y_1\,\ldots\,y_k\,x_1\,\ldots\,x_s \to K_i\,x_1\,\ldots\,x_s$ (whose right side already has application depth 2; the additional variables $x_1, \ldots, x_s$ are added to ensure that the type is o). By repeating such a replacement for every "bad" subterm of every rule, we clearly obtain a grammar in a simple form.

▶ **Lemma 4.1.** *Let $\mathcal{G}'$ be the grammar in a simple form obtained by the above simplification procedure from a grammar $\mathcal{G}$. Then $\mathsf{ord}(\mathcal{G}') = \mathsf{ord}(\mathcal{G})$, and $A_{\mathcal{G}'} \leq 2A_{\mathcal{G}}$, and $|\mathcal{G}'| = \mathcal{O}(A_{\mathcal{G}} \cdot |\mathcal{G}|)$. The procedure can be performed in time linear in its output size.*

Proof. The parts about the order and about the running time are obvious.

Types of nonterminals originating from $\mathcal{G}$ remain unchanged. The type of a fresh nonterminal Y introduced in the procedure is of the form $\alpha_1 \to \cdots \to \alpha_k \to \beta_1 \to \cdots \to \beta_s \to \mathsf{o}$,

where all α_i and β_i are types present also in $\mathcal{G}$. The arity of the whole type is $k+s$, where k is the arity of the original nonterminal X (hence it is bounded by $A_{\mathcal{G}}$), and s is bounded by the arity of the type of K_i (hence also by $A_{\mathcal{G}}$).

In order to bound the size of the resulting grammar, notice that the considered replacement is performed at most once for every subterm of the right side of every rule, hence the number of replacements is bounded by $|\mathcal{G}|$. Each such a replacement increases the size of the grammar by at most $\mathcal{O}(A_{\mathcal{G}})$. ◀

▶ **Lemma 4.2.** *For every grammar $\mathcal{G}$ in a simple form, the grammar $\mathcal{G}^\dagger$ (i.e., the result of the order-reducing transformation) is also in a simple form, and* $\mathsf{ord}(\mathcal{G}^\dagger) = \max(0, \mathsf{ord}(\mathcal{G})-1)$, *and $A_{\mathcal{G}^\dagger} \leq A_{\mathcal{G}} \cdot 2^{A_{\mathcal{G}}}$, and $|\mathcal{G}^\dagger| = \mathcal{O}(|\mathcal{G}| \cdot 2^{5 \cdot A_{\mathcal{G}}})$. Moreover, the transformation can be performed in time linear in its output size.*

Proof. The part about the running time is obvious. It is also easy to see by induction that $\mathsf{ord}(\alpha^\dagger) = \max(0, \mathsf{ord}(\alpha) - 1)$. It follows that the order of the grammar satisfies the same equality, because nonterminals of $\mathcal{G}^\dagger$ have type $\alpha^\dagger$ for α being the type of a corresponding nonterminal of $\mathcal{G}$.

Recall that in the type $\alpha^\dagger$ obtained from $\alpha = (\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o})$, every α_i either disappears or becomes (transformed and) repeated $2^{\mathsf{gar}(\alpha_i)}$ times, that is, at most $2^{A_{\mathcal{G}}}$ times. This implies the inequality concerning $A_{\mathcal{G}^\dagger}$.

Every compound application can be written as $f\, K_1 \dots K_k\, L_1 \dots L_\ell$, where f is a nonterminal or a variable, and $\ell = \mathsf{gar}(f)$. In such a term, every K_i (after transforming) becomes repeated $2^{\mathsf{gar}(K_i)}$ times, that is, at most $2^{A_{\mathcal{G}}}$ times. Then, for every L_i we duplicate the outcome and we append a small prefix; this duplication happens ℓ times, that is, at most $A_{\mathcal{G}}$ times. In consequence, we easily see by induction that while transforming a term of application depth d, its size gets multiplicated by at most $O(2^{2d \cdot A_{\mathcal{G}}})$. Moreover, every nonterminal X is repeated $2^{\mathsf{gar}(X)}$ times, that is, at most $2^{A_{\mathcal{G}}}$ times. Because the application depth of right sides of rules is at most 2, this bounds the size of the new grammar by $\mathcal{O}(|\mathcal{G}| \cdot 2^{5 \cdot A_{\mathcal{G}}})$.

Looking again at the above description of the transformation, we can notice that the application depth cannot grow; in consequence the property of being in a simple form is preserved. ◀

Thus, if we want to check nonemptiness of a grammar $\mathcal{G}$ of order n, we can first convert it to a simple form, and then apply the order-reducing transformation n times. This gives us a grammar of order 0, whose nonemptiness can be checked in linear time. By Lemmata 4.1 and 4.2, the whole algorithm works in time n-fold exponential in $A_{\mathcal{G}}$ and linear in $|\mathcal{G}|$.

If the original grammar $\mathcal{G}$ generates a language of words, we can start by applying the polynomial-time transformation of Asada and Kobayashi [2], which converts $\mathcal{G}$ into an equivalent grammar of order $n-1$ (generating a language of trees); then we can continue as above. Because their transformation is also linear in $|\mathcal{G}|$, and increases the arity only quadratically, in this case we obtain an algorithm working in time $(n-1)$-fold exponential in $A_{\mathcal{G}}$ and linear in $|\mathcal{G}|$.

5 Correctness

In this section we finish a proof of Theorem 3.1 by showing that the grammar $\mathcal{G}^\dagger$ resulting from transforming a grammar $\mathcal{G}$ is convergent if and only if the original grammar $\mathcal{G}$ is convergent. This proof is also formalised in the proof assistant Coq, and available at GitHub (https://github.com/pparys/ho-transform-sbs). The strategy of our proof is similar as in

Asada and Kobayashi [2]. Namely, we first show that reductions performed by $\mathcal{G}$ can be reordered, so that we can postpone substituting for (trailing) variables of order 0. To store such postponed substitutions, called *explicit substitutions*, we introduce *extended terms*. Then, we show that such reordered reductions in $\mathcal{G}$ are in a direct correspondence with reductions in $\mathcal{G}^\dagger$.[1]

Extended terms. In the sequel, terms defined previously are sometimes called non-extended terms, in order to distinguish them from extended terms defined below. Having a finite set of typed nonterminals $\mathcal{X}$, and a finite set $\mathcal{Z}$ of variables of type o, *extended terms* over $(\mathcal{X}, \mathcal{Z})$ are defined by induction:

- if $z \notin \mathcal{Z}$ is a variable of type o, and E is an extended term over $(\mathcal{X}, \mathcal{Z} \uplus \{z\})$, and L is a non-extended term of type o over $(\mathcal{X}, \mathcal{Z})$, then $E\langle L/z\rangle$ is an extended term over $(\mathcal{X}, \mathcal{Z})$;
- every non-extended term of type o over $(\mathcal{X}, \mathcal{Z})$ is an extended term over $(\mathcal{X}, \mathcal{Z})$.

The construction of the form $E\langle L/z\rangle$ is called an *explicit substitution*. Intuitively, it denotes the term obtained by substituting L for z in E. Notice that the variable z being free in E becomes bound in $E\langle L/z\rangle$, and that explicit substitutions are allowed only for the ground type o.

Of course a (non-extended or extended) term over $(\mathcal{X}, \mathcal{Z})$ can be also seen as a term over $(\mathcal{X}, \mathcal{Z}')$, where $\mathcal{Z}' \supseteq \mathcal{Z}$. In the sequel, such extending of the set of variables is often performed implicitly.

Having a grammar $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$, for every set $\mathcal{Z}$ of variables of type o we define an *ext-reduction* relation $\rightsquigarrow_{\mathcal{G}}$ between extended terms over $(\mathcal{X}, \mathcal{Z})$ and sets of such terms, as the least relation such that

(1) $X\,K_1\,\ldots\,K_k\,L_1\,\ldots\,L_\ell \rightsquigarrow_{\mathcal{G}} \{R[K_1/y_1, \ldots, K_k/y_k, z'_1/z_1, \ldots, z'_\ell/z_\ell]\langle L_1/z'_1\rangle \ldots \langle L_\ell/z'_\ell\rangle\}$ if $\ell = \mathsf{gar}(X)$, and $\mathcal{R}(X) = (X\,y_1\,\ldots\,y_k\,z_1\,\ldots\,z_\ell \to R)$, and $z'_1, \ldots, z'_\ell$ are fresh variables of type o not appearing in $\mathcal{Z}$,

(2) $\bullet\langle K_1, \ldots, K_k\rangle \rightsquigarrow_{\mathcal{G}} \{K_1, \ldots, K_k\}$,

(3) $\oplus\langle K_1, \ldots, K_k\rangle \rightsquigarrow_{\mathcal{G}} \{K_i\}$ for every $i \in [k]$,

(4) $z\langle L/z\rangle \rightsquigarrow_{\mathcal{G}} \{L\}$,

(5) $z'\langle L/z\rangle \rightsquigarrow_{\mathcal{G}} \{z'\}$ if $z' \neq z$, and

(6) $E\langle L/z\rangle \rightsquigarrow_{\mathcal{G}} \{F\langle L/z\rangle \mid F \in \mathcal{F}\}$ whenever $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$.

We say that an extended term E over $(\mathcal{X}, \emptyset)$ is *$\mathcal{G}$-ext-convergent* if $E \longrightarrow_{\mathcal{G}} \mathcal{F}$ for some set $\mathcal{F}$ of $\mathcal{G}$-ext-convergent extended terms. The grammar $\mathcal{G}$ is *ext-convergent* if its starting nonterminal X_0 is $\mathcal{G}$-ext-convergent.

There is an "expand" function from extended terms to non-extended terms, which performs all the explicit substitutions written in front of an extended term:

$$\exp(K\langle L_1/z_1\rangle \ldots \langle L_\ell/z_\ell\rangle) = K[L_1/z_1] \ldots [L_\ell/z_\ell].$$

We also write $\exp(\mathcal{F})$ for $\{\exp(F) \mid F \in \mathcal{F}\}$ (where $\mathcal{F}$ is a set of extended terms). The following lemma, saying that we can consider ext-convergence instead of convergence, can be proved in a standard way (actually, Asada and Kobayashi have a very similar lemma [2, Lemma 18]); a proof can be found in the full version of the paper (Appendix A).

[1] Asada and Kobayashi have an additional step in their proof, namely a reduction to the case of recursion-free grammars. This step turns out to be redundant, at least in the case of our transformation.

▶ **Lemma 5.1.** *Let $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ be a grammar. An extended term E over $(\mathcal{X}, \emptyset)$ is $\mathcal{G}$-ext-convergent if and only if $\exp(E)$ is $\mathcal{G}$-convergent. In particular $\mathcal{G}$ is ext-convergent if and only if it is convergent.*

We extend the transformation function to extended terms, by adding the following rule, where $E\langle L/z\rangle$ is an extended term over $(\mathcal{X}, \mathcal{Z})$, and $Z \in 2^{\mathcal{Z}}$ (the first argument is always $\emptyset$, because all extended terms are of type o):

(9) $\mathsf{tr}(\emptyset, Z, E\langle L/z\rangle) = \oplus\langle \mathsf{tr}(\emptyset, Z[z \mapsto 0], E), \bullet\langle \mathsf{tr}(\emptyset, Z[z \mapsto 1], E), \mathsf{tr}(\emptyset, Z, L)\rangle\rangle$.

Between ext-convergence and convergence of $\mathcal{G}^\dagger$. Once we know that convergence and ext-convergence of $\mathcal{G}$ are equivalent (cf. Lemma 5.1), it remains to prove that ext-convergence of $\mathcal{G}$ is equivalent to convergence of $\mathcal{G}^\dagger$, which is the subject of Lemma 5.2:

▶ **Lemma 5.2.** *Let $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ be a grammar. An extended term E over $(\mathcal{X}, \emptyset)$ is $\mathcal{G}$-ext-convergent if and only if $\mathsf{tr}(\emptyset, \emptyset, E)$ is $\mathcal{G}^\dagger$-convergent. In particular $\mathcal{G}$ is ext-convergent if and only if $\mathcal{G}^\dagger$ is convergent.*

The remaining part of this section is devoted to a proof of this lemma. Fix a grammar $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$. Of course the second part (concerning the grammars) follows from the first part (concerning an extended term) applied to the starting nonterminal X_0. It is thus enough to prove the first part. We start with the left-to-right direction (i.e., from $\mathcal{G}$-ext-convergence of E to $\mathcal{G}^\dagger$-convergence of $\mathsf{tr}(\emptyset, \emptyset, E)$). We need two simple auxiliary lemmata. The first of them says that the tr function commutes with substitution:

▶ **Lemma 5.3.** *Let $R[K_1/y_1, \ldots, K_k/y_k]$ be a term over $(\mathcal{X}, \mathcal{Z})$, let $A \in 2^{[\mathsf{gar}(R)]}$, and let $Z \in 2^{\mathcal{Z}}$. Then*

$$\mathsf{tr}(A, Z, R[K_1/y_1, \ldots, K_k/y_k]) = (\mathsf{tr}(A, Z, R))[\mathsf{tr}(B, Z, K_i)/y^\dagger_{i,B} \mid i \in [k], B \in 2^{[\mathsf{gar}(K_i)]}].$$

Proof. A straightforward induction on the structure of R. ◀

The second lemma says that by increasing values of the function Z we can make the transformed term only more convergent:

▶ **Lemma 5.4.** *Let E be an extended term over $(\mathcal{X}, \mathcal{Z} \uplus \{z\})$, and let $Z \in 2^{\mathcal{Z}}$. If $\mathsf{tr}(\emptyset, Z[z \mapsto 0], E)$ is $\mathcal{G}^\dagger$-convergent, then also $\mathsf{tr}(\emptyset, Z[z \mapsto 1], E)$ is $\mathcal{G}^\dagger$-convergent.*

Proof. Denote $P^0 = \mathsf{tr}(\emptyset, Z[z \mapsto 0], E)$ and $P^1 = \mathsf{tr}(\emptyset, Z[z \mapsto 1], E)$. Tracing the rules of the transformation function, we can see that P^0 and P^1 are created in the same way, with the exception that occurrences of z in E are transformed to Ω in P^0, and to $\bullet$ in P^1. Thus, P^1 can be obtained from P^0 by replacing some occurrences of Ω to $\bullet$. We know that P^0 is $\mathcal{G}^\dagger$-convergent, which means that it can be rewritten using the $\longrightarrow_{\mathcal{G}}$ relation until reaching empty sets. Moreover, the subterms Ω (which are present in P^0, but not in P^1) cannot be reached during this rewriting, because Ω is not $\mathcal{G}^\dagger$-convergent. Thus, P^1 can be rewritten in exactly the same way as P^0, so it is also $\mathcal{G}^\dagger$-convergent. ◀

The next lemma shows how ext-reductions of $\mathcal{G}$ are reflected in $\mathcal{G}^\dagger$:

▶ **Lemma 5.5.** *Let E be an extended term over $(\mathcal{X}, \mathcal{Z})$ and let $Z \in 2^{\mathcal{Z}}$. If $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ and $\mathsf{tr}(\emptyset, Z, F)$ is $\mathcal{G}^\dagger$-convergent for every $F \in \mathcal{F}$, then $\mathsf{tr}(\emptyset, Z, E)$ is also $\mathcal{G}^\dagger$-convergent.*

Proof. Induction on the definition of $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$. We analyze particular cases appearing in the definition. Missing details are given in the full version of the paper (Appendix B).

In Case (1) E consists of an application of arguments to some nonterminal X. For simplicity of presentation, suppose that X has two arguments: y of positive order, and z of order 0 (the general case is handled in the full version of the paper). Then

$$E = X\,K\,L, \qquad \text{and} \qquad \mathcal{F} = \{F\} \qquad \text{for} \qquad F = R[K/y, z'/z]\langle L/z'\rangle,$$

where $\mathcal{R}(X) = (X\,y\,z \to R)$ and z' is a fresh variable of type o not appearing in $\mathcal{Z}$. For $j \in \{0,1\}$ let

$$P^j = \mathsf{tr}([1 \mapsto j], Z, X\,K), \qquad \text{and} \qquad Q^j = \mathsf{tr}(\emptyset, Z[z' \mapsto j], R[K/y, z'/z]).$$

First, we prove that $P^j \longrightarrow_{\mathcal{G}^\dagger} \{Q^j\}$. By definition we have that

$$P^j = X^\dagger_{[1\mapsto j]}\,(\mathsf{tr}(B, Z, K))_{B \in 2^{[\mathsf{gar}(K)]}},$$

and by Lemma 5.3 we have that

$$\begin{aligned} Q^j &= \mathsf{tr}(\emptyset, Z[z' \mapsto j], R[z'/z])[\mathsf{tr}(B, Z[z' \mapsto j], K)/y^\dagger_B \mid B \in 2^{[\mathsf{gar}(K)]}] \\ &= \mathsf{tr}(\emptyset, [z \mapsto j], R)[\mathsf{tr}(B, Z, K)/y^\dagger_B \mid B \in 2^{[\mathsf{gar}(K)]}]], \end{aligned}$$

where the second equality holds because the z' does not appear in K and the variables from $\mathrm{dom}(Z)$ do not appear in R. Recalling that the rule for $X^\dagger_A$ is

$$X^\dagger_{[1\mapsto j]}\,(y^\dagger_B)_{B \in 2^{[\mathsf{gar}(y)]}} \to \mathsf{tr}(\emptyset, [z \mapsto j], R),$$

we immediately see that indeed $P^j \longrightarrow_{\mathcal{G}^\dagger} \{Q^j\}$. Having this, we recall that

$$\mathsf{tr}(\emptyset, Z, E) = \oplus\langle P^0, \bullet\langle P^1, L'\rangle\rangle \qquad \text{and} \qquad \mathsf{tr}(\emptyset, Z, F) = \oplus\langle Q^0, \bullet\langle Q^1, L'\rangle\rangle \tag{1}$$

for appropriate L' (obtained by transforming L). Recall that, by definition, a term M is $\mathcal{G}^\dagger$-convergent if and only if $M \longrightarrow_{\mathcal{G}^\dagger} \mathcal{N}$ for some set $\mathcal{N}$ of $\mathcal{G}^\dagger$-convergent terms. Thus, the only way why $\mathsf{tr}(\emptyset, Z, F)$ can be $\mathcal{G}^\dagger$-convergent (which holds by assumption) is that either Q^0 is $\mathcal{G}^\dagger$-convergent, or both Q^1 and L' are $\mathcal{G}^\dagger$-convergent. Because of the reduction $P^j \longrightarrow_{\mathcal{G}^\dagger} \{Q^j\}$ we have that either P^0 is $\mathcal{G}^\dagger$-convergent, or both P^1 and L' are $\mathcal{G}^\dagger$-convergent, which implies that $\mathsf{tr}(\emptyset, Z, E)$ is $\mathcal{G}^\dagger$-convergent.

In Cases (2) and (3), when $E = \bullet\langle K_1, \ldots, K_k\rangle$ or $E = \oplus\langle K_1, \ldots, K_k\rangle$, we have a reduction from $\mathsf{tr}(\emptyset, Z, E)$ to $\{\mathsf{tr}(\emptyset, Z, F) \mid F \in \mathcal{F}\}$, because tr distributes over $\bullet\langle\ldots\rangle$ and $\oplus\langle\ldots\rangle$. In Cases (4) and (5) (elimination of explicit substitution) we also have similar reductions.

Finally, in Case (6) we have that

$$E = E_0\langle L/z\rangle, \qquad \mathcal{F} = \{E_1\langle L/z\rangle, \ldots, E_k\langle L/z\rangle\}, \qquad \text{and} \qquad E_0 \rightsquigarrow_{\mathcal{G}} \{E_1, \ldots, E_k\}.$$

By definition, for every $i \in \{0, \ldots, k\}$ we have that

$$\begin{aligned} &\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle) = \oplus\langle P^0_i, \bullet\langle P^1_i, L'\rangle\rangle, \qquad \text{where} \\ &P^0_i = \mathsf{tr}(\emptyset, Z[z \mapsto 0], E_i), \qquad P^1_i = \mathsf{tr}(\emptyset, Z[z \mapsto 1], E_i), \qquad L' = \mathsf{tr}(\emptyset, Z, L). \end{aligned} \tag{2}$$

Thus, $\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle)$ is $\mathcal{G}^\dagger$-convergent if and only if either P^0_i is $\mathcal{G}^\dagger$-convergent, or both P^1_i and L' are $\mathcal{G}^\dagger$-convergent. By assumption this is the case for all $i \in [k]$, and we have to prove this for $i = 0$. If for every $i \in [k]$ we have the former case (i.e., P^0_i is $\mathcal{G}^\dagger$-convergent), by the

induction hypothesis (used with the function $Z[z \mapsto 0]$) we have that P_0^0 is $\mathcal{G}^\dagger$-convergent, and we are done. In the opposite case, for some $i \in [k]$ (but for at least one of them) we have that both P_i^1 and L' are $\mathcal{G}^\dagger$-convergent, and for the remaining $i \in [k]$ we have that P_i^0 is $\mathcal{G}^\dagger$-convergent. Using Lemma 5.4 we deduce that if P_i^0 is $\mathcal{G}^\dagger$-convergent, then also P_i^1 is $\mathcal{G}^\dagger$-convergent. Thus actually P_i^1 is $\mathcal{G}^\dagger$-convergent for every $i \in [k]$, and additionally L' is $\mathcal{G}^\dagger$-convergent. By the induction hypothesis (used with the function $Z[z \mapsto 1]$) we have that P_0^1 is $\mathcal{G}^\dagger$-convergent, and we are also done. ◀

We can now conclude with the left-to-right direction of Lemma 5.2:

▶ **Lemma 5.6.** *Let E be an extended term over $(\mathcal{X}, \emptyset)$. If E is $\mathcal{G}$-ext-convergent, then $\mathsf{tr}(\emptyset, \emptyset, E)$ is $\mathcal{G}^\dagger$-convergent.*

Proof. Induction on the fact that E is $\mathcal{G}$-ext-convergent. Because E is $\mathcal{G}$-ext-convergent, $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ for some set $\mathcal{F}$ of $\mathcal{G}$-ext-convergent extended terms, for which we can apply the induction hypothesis. The induction hypothesis says that $\mathsf{tr}(\emptyset, \emptyset, F)$ is $\mathcal{G}^\dagger$-convergent for every $F \in \mathcal{F}$. In such a situation Lemma 5.5 implies that $\mathsf{tr}(\emptyset, \emptyset, E)$ is also $\mathcal{G}^\dagger$-convergent, as required. ◀

For a proof in the opposite direction we need the following definition. We say that a term M *$\mathcal{G}^\dagger$-convergent in n steps* if $M \longrightarrow_{\mathcal{G}^\dagger} \{N_1, \ldots, N_k\}$, and every N_i is $\mathcal{G}^\dagger$-convergent in n_i steps, and $n = 1 + n_1 + \cdots + n_k$ (i.e., we count 1 for the above reduction, and we sum the numbers of steps needed to reduce all N_i). Clearly a term M is $\mathcal{G}^\dagger$-convergent if and only if it is $\mathcal{G}^\dagger$-convergent in n steps for some $n \in \mathbb{N}$. Notice that the number n is not determined by M (i.e., that the same term M can be $\mathcal{G}^\dagger$-convergent in n steps for multiple values of n). We can now state the converse of Lemma 5.5:

▶ **Lemma 5.7.** *Let E be an extended term over $(\mathcal{X}, \mathcal{Z})$ and let $Z \in 2^{\mathcal{Z}}$. If $\mathsf{tr}(\emptyset, Z, E)$ is $\mathcal{G}^\dagger$-convergent in n steps and E is not a variable, then there exists a set $\mathcal{F}$ of extended terms such that $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ and $\mathsf{tr}(\emptyset, Z, F)$ is $\mathcal{G}^\dagger$-convergent in less than n steps for every $F \in \mathcal{F}$.*

Proof. Induction on the number of explicit substitutions in E. Depending on the shape of E, we have several cases. Missing details are given in the full version of the paper (Appendix C).

One case is E consists of a nonterminal X with some arguments applied. For simplicity of presentation, we again suppose that X has two arguments: y of positive order, and z of order 0. Thus, E is of the form $E = X\,K\,L$. Let $X\,y\,z \to R$ be the rule for X, and let z' be a fresh variable of type o not appearing in $\mathcal{Z}$. In such a situation, taking $F = R[K/y, z'/z]\langle L/z'\rangle$ we have that $E \rightsquigarrow_{\mathcal{G}} \{F\}$. Recall the terms P^j and Q^j (for $j \in \{0,1\}$) from the proof of Lemma 5.5. In that proof we have observed that $P^j \longrightarrow_{\mathcal{G}^\dagger} \{Q^j\}$. But clearly this is the only way how P^j can reduce, so if P^j is $\mathcal{G}^\dagger$-convergent in n_j steps, then necessarily Q^j is $\mathcal{G}^\dagger$-convergent in $n_j - 1$ steps. By Equalities (1) we have that if $\mathsf{tr}(\emptyset, Z, E)$ is $\mathcal{G}^\dagger$-convergent in n steps, then either P^0 is $\mathcal{G}^\dagger$-convergent in $n_0 = n - 1$ steps, or both P^1 and L' are $\mathcal{G}^\dagger$-convergent in, respectively, n_1 and $n - n_1 - 2$ steps, for some $n_1 \in \mathbb{N}$. In the former case, Q^0 is $\mathcal{G}^\dagger$-convergent in $n_0 - 1 = n - 2$ steps, so $\mathsf{tr}(\emptyset, Z, F)$ is $\mathcal{G}^\dagger$-convergent in $n - 1$ steps, and we are done. In the latter case, Q^1 is $\mathcal{G}^\dagger$-convergent in $n_1 - 1$ steps, so $\mathsf{tr}(\emptyset, Z, F)$ is $\mathcal{G}^\dagger$-convergent in $(n_1 - 1) + (n - n_1 - 2) + 2 = n - 1$ steps, and we are done again.

Notice that we do not have a similar case for a variable with some arguments applied, because the whole E is not a variable, and because (by definition of an extended term) all free variables of E are of type o.

The cases of $E = \bullet\langle K_1, \ldots, K_k\rangle$ and $E = \oplus\langle K_1, \ldots, K_k\rangle$ are straightforward.

It remains to assume that E is an explicit substitution. If $E = z\langle L/z\rangle$, we should take $\mathcal{F} = \{L\}$, and if $E = z'\langle L/z\rangle$ for $z' \neq z$, we should take $\mathcal{F} = \{z'\}$ (in these two subcases we cannot use the induction assumption, because it does not work for an extended term being a single variable). Otherwise $E = E_0\langle L/z\rangle$, where E_0 is not a variable. Recall that $\mathsf{tr}(\emptyset, Z, E) = \oplus\langle P_0^0, \bullet\langle P_0^1, L'\rangle\rangle$ for P_0^0, P_0^1, L' as in the proof of Lemma 5.5. By assumption $\mathsf{tr}(\emptyset, Z, E)$ is $\mathcal{G}^\dagger$-convergent in n steps, so either P_0^0 is $\mathcal{G}^\dagger$-convergent in $n' = n-1$ steps, or both P_0^1 and L' are $\mathcal{G}^\dagger$-convergent in, respectively, n' and $n - n' - 2$ steps, for some $n' \in \mathbb{N}$. Let $j = 0$ in the former case and $j = 1$ in the latter case. The induction hypothesis gives us a set $\{E_1, \ldots, E_k\}$ such that $E_0 \rightsquigarrow_{\mathcal{G}} \{E_1, \ldots, E_k\}$ and $\mathsf{tr}(\emptyset, Z[z \mapsto j], E_i)$ is $\mathcal{G}^\dagger$-convergent in less than n' steps for every $i \in [k]$. We then take

$$\mathcal{F} = \{E_1\langle L/z\rangle, \ldots, E_k\langle L/z\rangle\}.$$

Equality (2) holds now for all $i \in \{0, \ldots, k\}$. For $j = 0$ we use that the fact that $\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle) \longrightarrow_{\mathcal{G}^\dagger} \{P_i^0\}$, which implies that $\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle)$ is $\mathcal{G}^\dagger$-convergent in less than $n'+1 = n$ steps, as required. For $j = 1$ we use that the fact that $\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle) \longrightarrow_{\mathcal{G}^\dagger} \{\bullet\langle P_i^1, L'\rangle\}$ and $\bullet\langle P_i^1, L'\rangle \longrightarrow_{\mathcal{G}^\dagger} \{P_i^1, L'\}$, which implies that $\mathsf{tr}(\emptyset, Z, E_i\langle L/z\rangle)$ is $\mathcal{G}^\dagger$-convergent in less than $n' + (n - n' - 2) + 2 = n$ steps, as required. ◀

The next lemma finishes the proof of Lemma 5.2, and thus the proof of correctness of our transformation:

▶ **Lemma 5.8.** *Let E be an extended term over $(\mathcal{X}, \emptyset)$. If $\mathsf{tr}(\emptyset, \emptyset, E)$ is $\mathcal{G}^\dagger$-convergent then E is $\mathcal{G}$-ext-convergent.*

Proof. Induction on the (smallest) number n such that $\mathsf{tr}(\emptyset, \emptyset, E)$ is $\mathcal{G}^\dagger$-convergent in n steps. By assumption E is not a variable, because it is an extended term over $(\mathcal{X}, \emptyset)$ (no free variables). So, by Lemma 5.5 there exists a set $\mathcal{F}$ of extended terms such that $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ and $\mathsf{tr}(\emptyset, \emptyset, F)$ is $\mathcal{G}^\dagger$-convergent in less than n steps for every $F \in \mathcal{F}$. By the induction hypothesis every $F \in \mathcal{F}$ is $\mathcal{G}$-ext-convergent, so by definition also E is $\mathcal{G}$-ext-convergent. ◀

6 Conclusions

We have presented a new, simple algorithm checking whether a higher-order grammar generates a nonempty language. One may ask whether this algorithm can be used in practice. Of course the complexity n-EXPTIME for grammars of order n is unacceptably large (even if we take into account the fact that we are n-fold exponential only in the arity of types, not in the size of a grammar), but one has to recall that there exist tools solving the considered problem in such a complexity. The reason why these tools work is that the time spent by them on "easy" inputs is much smaller than the worst-case complexity (and many "typical inputs" are indeed easy). Unfortunately, this is not the case for our algorithm: the size of the grammar resulting from our transformation is always large, even if the original grammar generated a nonempty (or empty) language for some "easy reason". Thus, our algorithm is mainly of a theoretical interest.

The presented transformation preserves nonemptiness, and thus can be used to solve the nonemptiness problem for higher-order grammars. However, it seems feasible that other problems concerning higher-order grammars (higher-order recursion schemes), like model-checking against parity automata or the simultaneous unboundedness problem [7], can be solved using similar transformations. Developing such transformations is a possible direction for further work.

References

1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Log. Methods Comput. Sci.*, 3(3), 2007. doi:10.2168/LMCS-3(3:1)2007.

2 Kazuyuki Asada and Naoki Kobayashi. Size-preserving translations from order-(n+1) word grammars to order-n tree grammars. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 22:1–22:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.22.

3 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

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

5 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

6 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

7 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. *CoRR*, abs/1605.00371, 2016. arXiv:1605.00371.

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

9 Matthew Hague, Roland Meyer, Sebastian Muskalla, and Martin Zimmermann. Parity to safety in polynomial time for pushdown and collapsible pushdown systems. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.57.

10 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

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

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

13 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

14 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

15 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:9)2011.

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

17 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

18 Paweł Parys. Recursion schemes and the WMSO+U logic. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.53.

19 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

20 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

21 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

22 Taku Terao and Naoki Kobayashi. A ZDD-based efficient higher-order model checking algorithm. In Jacques Garrigue, editor, *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 354–371. Springer, 2014. doi:10.1007/978-3-319-12736-1_19.