


# Size-Preserving Translations from Order- $(n + 1)$ Word Grammars to Order- $n$ Tree Grammars

Kazuyuki Asada 

Tohoku University, Sendai, Japan  
<http://www.riec.tohoku.ac.jp/~asada/>  
asada@riec.tohoku.ac.jp

Naoki Kobayashi 

The University of Tokyo, Japan  
<http://www-kb.is.s.u-tokyo.ac.jp/~koba/>  
koba@kb.is.s.u-tokyo.ac.jp

---

## Abstract

Higher-order grammars have recently been studied actively in the context of automated verification of higher-order programs. Asada and Kobayashi have previously shown that, for any order- $(n + 1)$  word grammar, there exists an order- $n$  grammar whose frontier language coincides with the language generated by the word grammar. Their translation, however, blows up the size of the grammar, which inhibited complexity-preserving reductions from decision problems on word grammars to those on tree grammars. In this paper, we present a new translation from order- $(n + 1)$  word grammars to order- $n$  tree grammars that is size-preserving in the sense that the size of the output tree grammar is polynomial in the size of an input tree grammar. The new translation and its correctness proof are arguably much simpler than the previous translation and proof.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory

**Keywords and phrases** higher-order grammar, word language, tree language, complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSCD.2020.22

**Related Version** A full version of the paper is available at <http://www.riec.tohoku.ac.jp/~asada/papers/spWtoT.pdf>.

**Funding** This work was supported by JSPS Kakenhi 15H05706, 18K11156, and 20H00577.

**Acknowledgements** We would like to thank anonymous referees for useful comments.

## 1 Introduction

It is well known that there is a close relationship between context-free word languages and regular tree languages: for any regular tree language  $L$ , its frontier language, i.e., the word language consisting of the sequence of leaf symbols of each tree in  $L$ , is context-free; conversely, for any context-free language with no empty sequence, the set of parse trees is a regular tree language. Damm [6] has shown that this correspondence generalizes to *safe* higher-order languages: for any order- $n$  (safe) language  $L$ , there is a corresponding order- $(n + 1)$  word language that corresponds to the frontier language of  $L$ , and vice versa. Asada and Kobayashi [1] extended the result to *unsafe* higher-order languages. The results allow us to reduce a problem on order- $(n + 1)$  word languages (like linear-time property verification of order- $(n + 1)$  functional programs) to a problem on order- $n$  (but tree) languages. Such a reduction may be useful, since the cost of various problems on a higher-order language (such as HORS model checking) is usually in the tower of exponentials whose height is the order of the language.



© Kazuyuki Asada and Naoki Kobayashi;

licensed under Creative Commons License CC-BY

5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020).

Editor: Zena M. Ariola; Article No. 22; pp. 22:1–22:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Unfortunately, however, the translation of Asada and Kobayashi [1] from order- $(n + 1)$  word grammars to order- $n$  tree grammars (which is the hard direction) blows up the size of the grammar: the output tree grammar is hyper-exponential in the size of an input word grammar, which cancels off the benefit of reducing the order of the grammar. Also, both the translation and correctness proof were very complex. Clemente et al. [5] have shown another translation from order- $(n + 1)$  word grammars to order- $n$  tree grammars (as a component of their algorithm for the diagonal problem), but their translation also suffers from the same problem of blowing up the size of the grammar.

In the present paper, we present a much simpler translation from order- $(n + 1)$  word grammars to order- $n$  tree grammars, where the size of the output grammar is *polynomial* in the size of an input grammar.<sup>1</sup>

Some of the known results on the complexity of decision problems on higher-order languages follow immediately from our result. For example, Parys [12] has shown that the complexity of the diagonal problem is  $n$ -EXPTIME-complete for order- $n$  tree grammars and  $(n - 1)$ -EXPTIME-complete for order- $n$  word grammars. He separately discussed the tree and word cases, but using our result, the upper-bound for the word-case follows immediately from the tree case. (Note that the hardness follows from the translation of [1] in the opposite direction.) For another example, it is known that the inclusion problem between an order- $(n + 1)$  word language and a regular language (to which various program verification problems can be reduced) can be decided in  $n$ -EXPTIME in the size of the order- $(n + 1)$  word grammar; it is, for example, obtained as a corollary of Kobayashi and Ong’s result [10] that linear-time property model checking of order- $(n + 1)$  higher-order recursion schemes (HORS) is  $n$ -EXPTIME complete. Using our result, the upper-bound of the complexity of the inclusion problem can be obtained as an immediate corollary of the  $n$ -EXPTIME completeness of the modal  $\mu$ -calculus model checking of HORS [11] and our result.

Our new translation from word grammars to tree grammars is in a sense more elementary than the previous known translations [1, 5]. While the previous translations used intersection types, our new translation, which has been inspired from [3, 9], uses only simple types. The correctness proof of the new translation is also arguably much simpler than those of the previous translations.

**Related Work.** As explained above, a translation from order- $(n + 1)$  word grammars to order- $n$  tree grammars has first been shown by Damm [6, Theorem 7.17] for safe grammars. His translation does not seem to generalize to unsafe grammars (grammars without the so called “safety restriction” [7, 4]). Asada and Kobayashi [1] and Clemente et al. [5] have shown translations for unsafe grammars. As mentioned already, these previous translations are quite different from ours. Both of the previous translations [1, 5] used intersection types and replicated a term for each intersection type. Since the number of intersection types need for the translation of order- $k$  grammars is  $k$ -fold exponential, it was inevitable for the previous translations to blow up the size of the grammar. In contrast, our translation does not use intersection types, and only keeps track of how order-0 variables are used.

As for applications, Asada and Kobayashi’s translation [1] has been used to prove a pumping lemma for higher-order grammars *modulo* a certain conjecture [2]. Clemente et al. [5] used (an extension of) their translation to prove the decidability of the diagonal problem.

---

<sup>1</sup> More precisely, the transformation of Asada and Kobayashi [1] consists of two steps, a main step and an administrative step. We simplify only the main step; for applications discussed below, we do not need the administrative step.

Our translation has been inspired from related transformations developed in [3, 9]. Asada and Kobayashi [3] used a related transformation for order-3 grammars, which was used to prove a variant of the pumping lemma for higher-order grammars. Kobayashi et al. [9] used a similar transformation technique for probabilistic higher-order grammars called PHORS, to provide a fixpoint characterization of the termination probability of PHORS.

The rest of the paper is structured as follows. Section 2 reviews the definition of higher-order grammars, and states the main result. Section 3 gives the definition of the new transformation from order- $(n + 1)$  word grammars to order- $n$  tree grammars. Section 4 proves the complexity results, and discusses an application of the main result. Section 5 proves the correctness of the translation. Section 6 concludes the paper.

## 2 Preliminaries

We first review basic definitions on higher-order grammars [6, 1] in Section 2.1; our definitions given below basically follow those of [1], with slight modifications. We then state our main result in Section 2.2.

We often write  $\tilde{a}$  for a sequence  $a_1, \dots, a_n$ , and write  $|\tilde{a}|$  for the length  $n$  of the sequence.

### 2.1 Higher-order grammars

► **Definition 1** (types and terms). *The set of (simple) types, ranged over by  $\kappa$ , is given by:  $\kappa ::= \circ \mid \kappa_1 \rightarrow \kappa_2$ . The order, arity, and size of a type  $\kappa$ , written  $\text{ord}(\kappa)$ ,  $\text{ari}(\kappa)$ , and  $|\kappa|$ , are defined by*

$$\begin{array}{ll} \text{ord}(\circ) := 0 & \text{ord}(\kappa_1 \rightarrow \kappa_2) := \max(\text{ord}(\kappa_1) + 1, \text{ord}(\kappa_2)) \\ \text{ari}(\circ) := 0 & \text{ari}(\kappa_1 \rightarrow \kappa_2) := 1 + \text{ari}(\kappa_2) \\ |\circ| := 1 & |\kappa_1 \rightarrow \kappa_2| := |\kappa_1| + |\kappa_2| + 1. \end{array}$$

The type  $\circ$  describes trees, and  $\kappa_1 \rightarrow \kappa_2$  describes functions from  $\kappa_1$  to  $\kappa_2$ . The set of terms, ranged over by  $s, t, u, \dots$ , is defined by:

$$t ::= x \mid a t_1 \cdots t_k \mid t_1 t_2 \mid t_1 \oplus t_2 \mid \Omega.$$

Here,  $x$  ranges over a denumerable set of variables, and  $a$  over a set of constants (which represent tree constructors). We also use meta-variables  $y, z, F, G$  for variables. Variables and constants are also called non-terminals and terminals respectively. A ranked alphabet  $\Sigma$  is a map from a finite set of terminals to the set of natural numbers; we call  $\Sigma(a)$  the arity of a terminal  $a$ . We implicitly assume a ranked alphabet whose domain contains all terminals discussed, unless explicitly described. The term  $t_1 \oplus t_2$  denotes a non-deterministic choice between  $t_1$  and  $t_2$ , and  $\Omega$  denotes divergence. A term is called an applicative term if it contains neither  $\oplus$  nor  $\Omega$ .

A (simple) type environment  $\mathcal{K}$  is a sequence of type bindings of the form  $x : \kappa$ , where  $\mathcal{K}$  may contain at most one binding for each variable. The type judgment relation  $\mathcal{K} \vdash_{\text{ST}} t : \kappa$  is defined by the following rules.

$$\begin{array}{c} \frac{}{\mathcal{K}, x : \kappa, \mathcal{K}' \vdash_{\text{ST}} x : \kappa} \quad \frac{\Sigma(a) = k \quad \mathcal{K} \vdash_{\text{ST}} t_i : \circ \text{ (for each } i \in \{1, \dots, k\})}{\mathcal{K} \vdash_{\text{ST}} a t_1 \cdots t_k : \circ} \\ \frac{\mathcal{K} \vdash_{\text{ST}} t_1 : \kappa_2 \rightarrow \kappa \quad \mathcal{K} \vdash_{\text{ST}} t_2 : \kappa_2}{\mathcal{K} \vdash_{\text{ST}} t_1 t_2 : \kappa} \quad \frac{\mathcal{K} \vdash_{\text{ST}} t_1 : \circ \quad \mathcal{K} \vdash_{\text{ST}} t_2 : \circ}{\mathcal{K} \vdash_{\text{ST}} t_1 \oplus t_2 : \circ} \quad \frac{}{\mathcal{K} \vdash_{\text{ST}} \Omega : \circ} \end{array}$$

For a technical convenience, in the typing rule for constants, we require that a terminal must be fully applied; this does not restrict the expressive power of higher-order grammars introduced below. Note that, given  $\mathcal{K}$  and  $t$ , there exists at most one type  $\kappa$  such that  $\mathcal{K} \vdash_{\text{ST}} t : \kappa$ . We call  $\kappa$  the type of  $t$  (with respect to  $\mathcal{K}$ ). Henceforth, we consider only well-typed terms.

A term  $t$  is called ground (with respect to  $\mathcal{K}$ ) if  $\mathcal{K} \vdash_{\text{ST}} t : \circ$ , and  $t$  is called a (finite,  $\Sigma$ -ranked) tree if  $t$  is a closed ground applicative term consisting of only terminals. We write  $\mathbf{Tree}_\Sigma$  for the set of  $\Sigma$ -ranked trees, and use the meta-variable  $v$  for trees.

We define the size  $|t|$  of a term  $t$  by:  $|x| := 1$ ,  $|a t_1 \cdots t_k| := 1 + |t_1| + \cdots + |t_k|$ ,  $|s t| := |s| + |t| + 1$ ,  $|s \oplus t| := |s| + |t| + 1$ , and  $|\Omega| := 1$ .

► **Definition 2** (higher-order grammar). A higher-order grammar (or grammar for short) is a quadruple  $(\Sigma, \mathcal{N}, \mathcal{R}, t^\circ)$ , where

- $\Sigma$  is a ranked alphabet,
- $\mathcal{N}$  is a map from a finite set of non-terminals to their types,
- $\mathcal{R}$  is a finite set of rewriting rules of the form  $F x_1 \cdots x_\ell \rightarrow t$ , where: (i)  $t$  is a term, (ii)  $\mathcal{N}(F) = \kappa_1 \rightarrow \cdots \rightarrow \kappa_\ell \rightarrow \circ$ , (iii)  $\mathcal{N}, x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell \vdash_{\text{ST}} t : \circ$  holds, and (iv) in  $\mathcal{R}$  there is exactly one rule for each nonterminal,
- $t^\circ$  is a term called the start term, and  $\mathcal{N} \vdash_{\text{ST}} t^\circ : \circ$ .

The order of a grammar  $\mathcal{G}$  is defined as the largest order of the types of non-terminals (or 0 if  $\text{dom}(\mathcal{N})$  is empty). We define the size of  $\mathcal{N}$ , written as  $|\mathcal{N}|$ , by  $|\mathcal{N}| := \sum_{F \in \mathcal{N}} |\mathcal{N}(F)|$ , and also define the Curry-style and Church-style sizes of a grammar  $\mathcal{G}$ , written as  $|\mathcal{G}|_{\text{cy}}$  and  $|\mathcal{G}|_{\text{ch}}$ , by:  $|\mathcal{G}|_{\text{cy}} := |t^\circ| + \sum_{(F x_1 \cdots x_\ell \rightarrow t) \in \mathcal{R}} (|t| + \ell)$  and  $|\mathcal{G}|_{\text{ch}} := |\mathcal{G}|_{\text{cy}} + |\mathcal{N}|$ , respectively. We sometimes omit the subscript of  $|\mathcal{G}|_{\text{ch}}$  and write  $|\mathcal{G}|$ .  $|\mathcal{G}|_{\text{ch}}$  (rather than  $|\mathcal{G}|_{\text{cy}}$ ) is essentially the same as the definition of the grammar size in [12] (the results of [12] will be discussed in Section 4.2, as an application of our results).

The tree language  $\mathcal{L}(\mathcal{G})$  generated by a grammar  $\mathcal{G}$  is defined as follows. The set of evaluation contexts (ranged over by  $E$ ) is defined by the grammar:

$$E ::= [] \mid a t_1 \dots t_{i-1} E t_{i+1} \dots t_{\Sigma(a)} \quad (1 \leq i \leq \Sigma(a)) \mid E \oplus t \mid t \oplus E.$$

Below we consider only contexts  $E$  such that  $\mathcal{N}, \mathcal{K}, [] : \circ \vdash_{\text{ST}} E : \circ$ . For a grammar  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, t^\circ)$ , the rewriting relation  $\longrightarrow_{\mathcal{G}}$  is defined as follows:

$$\frac{(F \tilde{x} \rightarrow t) \in \mathcal{R}}{E[F \tilde{s}] \longrightarrow_{\mathcal{G}} E[\tilde{s}/\tilde{x}]t} \qquad \frac{}{E[a \tilde{s} (t_1 \oplus t_2) \tilde{u}] \longrightarrow_{\mathcal{G}} E[(a \tilde{s} t_1 \tilde{u}) \oplus (a \tilde{s} t_2 \tilde{u})]}$$

We write  $\longrightarrow_{\mathcal{G}}^*$  for the reflexive transitive closure of  $\longrightarrow_{\mathcal{G}}$ . We may omit the subscript  $\mathcal{G}$  and write  $\longrightarrow$  and  $\longrightarrow^*$ , if  $\mathcal{G}$  is clear from the context. We define the set of choice contexts (ranged over by  $C$ ) by:  $C ::= [] \mid C \oplus t \mid t \oplus C$ . For  $\mathcal{N} \vdash_{\text{ST}} t : \circ$ , we define  $\mathcal{L}(\mathcal{G}, t) := \{v \in \mathbf{Tree}_\Sigma \mid t \longrightarrow_{\mathcal{G}}^* C[v]\}$ , and  $\mathcal{L}(\mathcal{G}) := \mathcal{L}(\mathcal{G}, t^\circ)$ .

► **Remark 3.** In [1], we used a slightly different definition of a higher-order grammar: (i) The definition in the present paper uses a start term, while that in [1] uses a *start symbol* (i.e., a start term is restricted to some non-terminal  $S$ ). (ii) In the present paper  $\mathcal{R}$  is deterministic and total, and  $\oplus$  and  $\Omega$  may occur on the right hand side of each rule, while in [1]  $\mathcal{R}$  is not necessarily deterministic nor total, and neither  $\oplus$  nor  $\Omega$  may occur. The two styles of grammars can be mutually translated in an obvious manner.

The grammars defined above may also be viewed as generators of word languages.

► **Definition 4** (word alphabet / br-alphabet). A ranked alphabet  $\Sigma$  is called a word alphabet if it has a special nullary terminal  $\mathbf{e}$  and all the other terminals have arity 1. A grammar  $\mathcal{G}$  is called a word grammar if its alphabet is a word alphabet. For a tree  $v = a_1(\cdots(a_n \mathbf{e})\cdots)$  of a word grammar, we define  $\mathbf{word}(v) = a_1 \cdots a_n$ . The word language generated by a word grammar  $\mathcal{G}$ , written  $\mathcal{L}_w(\mathcal{G})$ , is  $\mathbf{word}(\mathcal{L}(\mathcal{G}))$ .

The frontier word of a tree  $v$ , written  $\mathbf{leaves}(v)$ , is the sequence of symbols in the leaves of  $v$ . It is defined inductively by:  $\mathbf{leaves}(a) = a$  when  $\Sigma(a) = 0$ , and  $\mathbf{leaves}(a v_1 \cdots v_k) = \mathbf{leaves}(v_1) \cdots \mathbf{leaves}(v_k)$  when  $\Sigma(a) = k > 0$ . We write  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{G})$  and  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{G}, t)$  for  $\mathbf{leaves}(\mathcal{L}(\mathcal{G}))$  and  $\mathbf{leaves}(\mathcal{L}(\mathcal{G}, t))$ , respectively, and call  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{G})$  the frontier language generated by  $\mathcal{G}$ .

A br-alphabet is a ranked alphabet such that it has a special binary constant  $\mathbf{br}$  and a special nullary constant  $\mathbf{e}$ , and the other constants are nullary. We call a grammar  $\mathcal{G}$  a br-grammar if its alphabet is a br-alphabet. Intuitively,  $\mathbf{br}$  and  $\mathbf{e}$  represent the concatenation of two frontier words and the empty word  $\varepsilon$  respectively. For a word  $w$ , we write  $w \uparrow_{\mathbf{e}}$  for the word obtained by removing all the occurrences of  $\mathbf{e}$  in  $w$ , and  $\mathcal{L} \uparrow_{\mathbf{e}}$  for  $\{w \uparrow_{\mathbf{e}} \mid w \in \mathcal{L}\}$ . We write  $s \approx t$  if  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{G}, s) \uparrow_{\mathbf{e}} = \mathcal{L}_{\mathbf{leaf}}(\mathcal{G}, t) \uparrow_{\mathbf{e}}$ .

For a word alphabet  $\Sigma$ , we define the br-alphabet of  $\Sigma$ , written  $\mathbf{br}(\Sigma)$ , by:  $\mathbf{br}(\Sigma) := \{\mathbf{e} \mapsto 0, \mathbf{br} \mapsto 2\} \cup \{a \mapsto 0 \mid \Sigma(a) = 1\}$ .

We note that the classes of order-0, order-1, and order-2 word languages coincide with those of regular, context-free, and indexed languages, respectively [13].

► **Example 5.** Consider the order-2 (word) grammar  $\mathcal{G}_1 = (\{\mathbf{a}:1, \mathbf{e}:0\}, \{S:\circ, F:(\circ \rightarrow \circ) \rightarrow (\circ \rightarrow \circ), T:(\circ \rightarrow \circ) \rightarrow (\circ \rightarrow \circ), A:\circ \rightarrow \circ\}, \mathcal{R}_1, S)$ , where  $\mathcal{R}_1$  consists of:

$$S \rightarrow F A \mathbf{e} \quad F f x \rightarrow (f x) \oplus (F(T f) x) \quad T f x \rightarrow f(f x) \quad A x \rightarrow \mathbf{a} x$$

$S$  is reduced, for example, as follows

$$\begin{aligned} S &\longrightarrow F A \mathbf{e} \longrightarrow C_1[F(T A) \mathbf{e}] \longrightarrow C_2[F(T(T A)) \mathbf{e}] \longrightarrow C_3[(T(T A)) \mathbf{e}] \\ &\longrightarrow C_3[T A(T A \mathbf{e})] \longrightarrow^* C_3[\mathbf{a}^4(\mathbf{e})] \end{aligned}$$

where  $C_1, C_2$ , and  $C_3$  are some appropriate choice contexts. The word language  $\mathcal{L}_w(\mathcal{G}_1)$  is  $\{\mathbf{a}^{2^n} \mid n \geq 0\}$ .

Consider the order-1 (tree) grammar  $\mathcal{G}_2 = (\{\mathbf{br}:2, \mathbf{a}:0, \mathbf{e}:0\}, \{S:\circ, F:\circ \rightarrow \circ, T:\circ \rightarrow \circ\}, \mathcal{R}_2, S)$ , where  $\mathcal{R}_2$  consists of:

$$S \rightarrow F \mathbf{a} \quad F f \rightarrow f \oplus (F(T f)) \quad T f \rightarrow \mathbf{br} f f.$$

The frontier language  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{G}_2)$  coincides with  $\mathcal{L}_w(\mathcal{G}_1)$  above. This (existence of an order-1 tree grammar corresponding to an order-2 word grammar) is not a coincidence, as stated in Theorem 6 below.

## 2.2 The main result

The following theorem states the main result of the present paper.

► **Theorem 6.** For any  $n \geq 0$ , there exist an effective translation  $\mathcal{T}_n$  from order- $(n+1)$  word grammars to order- $n$  br-grammars and a polynomial  $p_n$  such that, for any order- $(n+1)$  word grammar  $\mathcal{G}$ ,  $\mathcal{L}_{\mathbf{leaf}}(\mathcal{T}_n(\mathcal{G})) \uparrow_{\mathbf{e}} = \mathcal{L}_w(\mathcal{G})$  and  $|\mathcal{T}_n(\mathcal{G})| \leq p_n(|\mathcal{G}|)$ .

The theorem above follows from Theorems 12 and 13, given in Sections 4 and 5 respectively. Theorem 6 without the condition  $|\mathcal{T}_n(\mathcal{G})| \leq p_n(|\mathcal{G}|)$  has been proved in [1] (Theorem 7).

► Remark 7. Asada and Kobayashi [1] have also presented a post-processing transformation for removing  $\mathbf{e}$ , which also suffers from a hyper-exponential blow-up of the grammar size. We do not think that there exists a size-preserving transformation that achieves the removal of  $\mathbf{e}$ . Fortunately, however, the post-processing transformation is unnecessary for the applications discussed in the introduction.

### 3 The Transformation

A basic idea of our translation ( $\mathcal{T}_n$  in Theorem 6) to decrease the order of a grammar is to represent an order-1 word function as a tuple of order-0 terms, each of which represents the set of (the tree representations of) words that may be generated before a certain target term (such as an argument, a constant, or a variable) is encountered. For example, consider a term  $tu$  of type  $\mathfrak{o}$  where  $t$  has type  $\mathfrak{o} \rightarrow \mathfrak{o}$ . The set of words generated by  $tu$  is  $T_0 \cup (T_1 \cdot U_0)$ , where  $T_0 = \{w \mid tx \xrightarrow{*} w(\mathbf{e})\}$ ,  $T_1 = \{w \mid tx \xrightarrow{*} w(x)\}$  (for a fresh variable  $x$  of type  $\mathfrak{o}$ ), and  $U_0 = \{w \mid u \xrightarrow{*} w(\mathbf{e})\}$ . In other words,  $T_0$  is the set of words that are generated by  $t$  before  $\mathbf{e}$  is encountered (without using the argument), and  $T_1$  is the set of words that are generated before the argument is encountered. If we can convert  $t$  and  $u$  to  $t_0, t_1$ , and  $u_0$  that respectively generate tree representations of  $T_0, T_1$ , and  $U_0$ , then the whole term  $tu$  can be converted to  $t_0 \oplus (\mathbf{br} t_1 u_0)$  (where recall that the binary tree constructor  $\mathbf{br}$  plays the role of word concatenation). In this manner, the order-1 term  $t$  has been replaced by order-0 terms  $t_0$  and  $t_1$ . As a concrete example, consider an order-1 grammar:

$$S \rightarrow TU \quad Tx \rightarrow \mathbf{a} \mathbf{e} \oplus \mathbf{b} x \quad U \rightarrow \mathbf{c} \mathbf{e}.$$

Then it can be converted to the order-0 grammar:

$$S \rightarrow T_0 \oplus (\mathbf{br} T_1 U_0) \quad T_0 \rightarrow \mathbf{a} \quad T_1 \rightarrow \mathbf{b} \quad U_0 \rightarrow \mathbf{c}.$$

(In the actual translation below,  $\mathbf{e}$  should actually be passed around as a variable.)

For higher-order grammars, we apply the above transformation inductively (although a further twist is required as we explain later). For example, consider the grammar (which is a contrived version of Example 5).

$$S \rightarrow T A \mathbf{e} \quad T f x \rightarrow f(f x) \quad A x \rightarrow \mathbf{a} x.$$

Since the first argument  $f$  of  $T$  has type  $\mathfrak{o} \rightarrow \mathfrak{o}$ , it is replaced by an order-0 variable  $f_1$ , which is bound to a term that generates (the tree representation of) words generated by  $f$  before the argument is encountered.  $T$  is then replaced by an order-1 function  $T_1$  which, given  $f_1$ , generates words generated by  $T f x$  before  $x$  is encountered. The resulting grammar is thus:

$$S \rightarrow T_1 A_1 \quad T_1 f_1 \rightarrow \mathbf{br} f_1 f_1 \quad A_1 \rightarrow \mathbf{a}.$$

Note that  $f(f x)$  generates  $x$  only after the outer call of  $f$  uses the argument  $f x$ , and then the inner call of  $f$  uses the argument, hence the body  $\mathbf{br} f_1 f_1$  of  $T_1$ . Notice that the type of  $T_1$  is  $\mathfrak{o} \rightarrow \mathfrak{o}$ , which has order 1. In general, a function of type  $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$  (where  $\text{ord}(\kappa_k) > 0$ ) is converted to a tuple of functions of type  $\kappa_1^\dagger \rightarrow \cdots \rightarrow \kappa_k^\dagger \rightarrow \mathfrak{o}$ , where  $(\cdot)^\dagger$  represents recursive applications of the type conversion.

A further twist is required for higher-order cases. For example, consider the grammar:

$$S \rightarrow H \mathbf{e}(\mathbf{b} \mathbf{e}) \quad H x y \rightarrow F(G x y) \quad F g \rightarrow g A \quad G x y h \rightarrow (h x) \oplus (h(h y))$$

where the types of non-terminals are:

$$S : \circ \quad H : \circ \rightarrow \circ \rightarrow \circ \quad F : ((\circ \rightarrow \circ) \rightarrow \circ) \rightarrow \circ \quad G : \circ \rightarrow \circ \rightarrow (\circ \rightarrow \circ) \rightarrow \circ$$

(the rule and type of  $A$  are as before, hence omitted). Based on the idea above, the rule for  $S$  can be translated to:

$$S \rightarrow H_1 \oplus (\mathbf{br} H_2 \mathbf{b}),$$

where  $H_1$  ( $H_2$ , resp.) generates the words generated by  $Hxy$  before  $x$  ( $y$ , resp.) is encountered in the original grammar. But how can the rules for  $H_1$  and  $H_2$  be obtained from that of  $H$ ? The head symbol  $F$  of the body of  $H$  has no order-0 argument, so the idea explained above does not apply. We translate the rules for  $H$ ,  $F$ , and  $G$  to:

$$\begin{aligned} H_1 &\rightarrow F_0(G_0 \mathbf{e} \Omega) & H_2 &\rightarrow F_0(G_0 \Omega \mathbf{e}) & F_0 g_0 &\rightarrow g_0 A_1 \\ G_0 x_0 y_0 h_1 &\rightarrow (\mathbf{br} h_1 x_0) \oplus (\mathbf{br} h_1 (\mathbf{br} h_1 y_0)). \end{aligned}$$

Here,  $F_0 g_0$  generates the words generated by  $Fg$  before a certain target symbol (say,  $z$ ) is encountered, assuming that  $g_0 A_1$  generates the words generated by  $gA$  before  $z$  is encountered. In  $H_1$ , the argument  $g_0$  is set to  $G_0 \mathbf{e} \Omega$  (so that when  $x$  is reached in the original grammar, the empty word is generated, and when  $y$  is reached in the original grammar, no word is generated) since the target symbol is  $x$ , while in  $H_2$ ,  $g_0$  is set to  $G_0 \Omega \mathbf{e}$ . Similarly,  $G_0$  is a function to generate the words generated by  $Gxyh$  in the original grammar before a certain target symbol is encountered, assuming that the arguments  $x_0$  and  $y_0$  generate the words generated by  $x$  and  $y$  before the target is encountered.

The following is a variation of the example above:

$$\begin{aligned} S &\rightarrow H \mathbf{e} (\mathbf{b} \mathbf{e}) & Hxy &\rightarrow K(Gx)y & Kky &\rightarrow F(ky) & Fg &\rightarrow gA \\ Gxyh &\rightarrow (hx) \oplus (h(hy)) \end{aligned}$$

where  $K : (\circ \rightarrow (\circ \rightarrow \circ) \rightarrow \circ) \rightarrow \circ \rightarrow \circ$ . We have just introduced an auxiliary step to reduce  $Hxy$  to  $F(Gxy)$ , via  $K(Gx)y$ . It can be translated to:

$$\begin{aligned} S &\rightarrow H_1 \oplus (\mathbf{br} H_2 \mathbf{b}) & H_1 &\rightarrow K_0(G_0 \mathbf{e}) & H_2 &\rightarrow K_1(G_0 \Omega) & K_0 k_0 &\rightarrow F_0(k_0 \Omega) \\ K_1 k_1 &\rightarrow F_0(k_1 \mathbf{e}) & F_0 g_0 &\rightarrow g_0 A_1 & G_0 x_0 y_0 h_1 &\rightarrow (\mathbf{br} h_1 x_0) \oplus (\mathbf{br} h_1 (\mathbf{br} h_1 y_0)). \end{aligned}$$

Here,  $K_0$  is a function to generate the words generated by  $Kky$  before a certain target (embedded in  $k$ ) is encountered (thus,  $K_0(G_0 \mathbf{e})$  generates the words generated by  $K(Gx)y$  before the target  $x$  is encountered), whereas  $K_1$  is a function to generate the words generated by  $Kky$  before  $y$  is encountered. Different arguments  $G_0 \mathbf{e}$  and  $G_0 \Omega$  are, therefore, passed to  $K_0$  and  $K_1$ . In  $G_0 \mathbf{e}$ , the target is set to  $x$  (hence  $x$  has been replaced by  $\mathbf{e}$ ), whereas in  $G_0 \Omega$ , the target has not been set yet (and is later set to  $y$  when  $G_0 \Omega$  is passed to  $K_1$ ).

Note that the translations above have been chosen to clarify the essence of our transformation; they do not exactly match the actual translations defined below. Henceforth, we often write  $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ$  for  $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \rightarrow \circ^\ell \rightarrow \circ$  when either  $\text{ord}(\kappa_k) > 0$  or  $k = 0$ ; note that  $\kappa_1, \dots, \kappa_{k-1}$  (but not  $\kappa_k$ ) may be the ground type  $\circ$ . We abbreviate  $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ$  to  $\tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ$ , and define  $\text{gar}(\tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ) := \ell$ . We assume that a given word grammar is normalized to the form  $(\Sigma, \mathcal{N}, \mathcal{R}, S \mathbf{e})$ , where  $\mathcal{R}$  does not contain  $\mathbf{e}$ .

The discussions above suggest that for each term  $t$  of an order- $n$  type  $\tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ$ , with order-0 variables  $x_1, \dots, x_k$ , we need to consider the following tuple of order- $(n-1)$  terms in the target grammar:

$$(t_0, t_1, \dots, t_\ell, t_{\ell+1}, \dots, t_{\ell+k}, t_{\ell+k+1}).$$

Here, each element  $t_i$  of the tuple is a function used to generate (tree representations of) the words generated by  $t$  before a certain target is encountered, where the “target” is:

- the  $i$ -th ground-type argument (precisely, the  $i$ -th argument of the part  $\mathfrak{o}^\ell \rightarrow \mathfrak{o}$  in the type  $\tilde{\kappa} \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$  of  $t$ ) if  $1 \leq i \leq \ell$ ,
- $x_{i-\ell}$  if  $\ell + 1 \leq i \leq \ell + k$ ,
- set inside the term  $t$  if  $i = 0$ , and
- set later by the higher-order arguments of  $t$  if  $i = \ell + k + 1$ .

We formalize the translation for terms as a type-based transformation relation of the form:

$$\mathcal{K}; x_1, \dots, x_k \vdash_{\mathcal{N}} t : \tilde{\kappa} \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o} \rightsquigarrow (t_0, t_1, \dots, t_\ell, t_{\ell+1}, \dots, t_{\ell+k}, t_{\ell+k+1}).$$

where  $\mathcal{K}$  is a type environment and  $x_1, \dots, x_k$  are order-0 variables; we often omit the subscript of  $\vdash_{\mathcal{N}}$  and write  $\vdash$  if  $\mathcal{N}$  is clear from the context. The relation means that the term  $t$  of type  $\tilde{\kappa} \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$  is transformed to the tuple  $(t_0, t_1, \dots, t_\ell, t_{\ell+1}, \dots, t_{\ell+k}, t_{\ell+k+1})$ , where the role of each term is as described above. The relation is defined by the transformation rules given in Fig. 1.

The translation  $\mathcal{N}^\ddagger$  of the types of nonterminals used in (TR-GRAM) is defined as follows. We first define the translation of types by:

$$\begin{aligned} (\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})^\ddagger &:= \\ &(\kappa_1^\ddagger \rightarrow \dots \rightarrow \kappa_k^\ddagger \rightarrow \mathfrak{o}) \times (\kappa_1^{\ddagger'} \rightarrow \dots \rightarrow \kappa_k^{\ddagger'} \rightarrow \mathfrak{o})^\ell \times (\kappa_1^\ddagger \rightarrow \dots \rightarrow \kappa_k^\ddagger \rightarrow \mathfrak{o}), \\ (\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})^{\ddagger'} &:= (\kappa_1^{\ddagger'} \rightarrow \dots \rightarrow \kappa_k^{\ddagger'} \rightarrow \mathfrak{o})^\ell \times (\kappa_1^\ddagger \rightarrow \dots \rightarrow \kappa_k^\ddagger \rightarrow \mathfrak{o}), \\ (\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})^{\ddagger+m} &:= \\ &(\kappa_1^\ddagger \rightarrow \dots \rightarrow \kappa_k^\ddagger \rightarrow \mathfrak{o}) \times (\kappa_1^{\ddagger'} \rightarrow \dots \rightarrow \kappa_k^{\ddagger'} \rightarrow \mathfrak{o})^\ell \times (\kappa_1^\ddagger \rightarrow \dots \rightarrow \kappa_k^\ddagger \rightarrow \mathfrak{o})^{m+1} \end{aligned}$$

where  $m \geq -1$ . The translation of type environments is defined by:

$$(y_1 : \kappa_1, \dots, y_k : \kappa_k)^\ddagger := (y_{1,0}, \dots, y_{1, \text{gar}(\kappa_1)+1}) : \kappa_1^\ddagger, \dots, (y_{k,0}, \dots, y_{k, \text{gar}(\kappa_k)+1}) : \kappa_k^\ddagger.$$

Finally, we define  $\mathcal{N}^\ddagger$  by:

$$(F_1 : \kappa_1, \dots, F_k : \kappa_k)^\ddagger := (F_{1,0}, \dots, F_{1, \text{gar}(\kappa_1)}) : \kappa_1^{\ddagger-1}, \dots, (F_{k,0}, \dots, F_{k, \text{gar}(\kappa_k)}) : \kappa_k^{\ddagger-1}.$$

As in (TR-RULE), we translate each rule  $F y_1 \dots y_m x_1 \dots x_k \rightarrow t$  with  $\mathcal{N}(F) = \kappa_1 \rightarrow \dots \rightarrow \kappa_m \Rightarrow \mathfrak{o}^k \rightarrow \mathfrak{o}$  by the relation:

$$y_1 : \kappa_1, \dots, y_m : \kappa_m; x_1, \dots, x_k : \mathfrak{o} \vdash_{\mathcal{N}} t : \mathfrak{o} \rightsquigarrow (t_0, t_1, \dots, t_k, t_{k+1}).$$

Note that every  $t$  being transformed never contains  $\mathfrak{e}$  since  $\mathcal{R}$  does not contain  $\mathfrak{e}$ .

We now explain some of the key rules. There are two rules for variables: (TR-VARG) for ground type variables  $x_1, \dots, x_k$ , and (TR-VAR) for variables bound in  $\mathcal{K}$  (note that some of them also may have ground type  $\mathfrak{o}$ ). In (TR-VARG), the  $(i+1)$ -th component of the output should represent the words generated before  $x_i$  is encountered; since  $x_i$  is immediately encountered, the component is set to  $\mathfrak{e}$ . The other components are set to  $\Omega$  (which generates no word), since no other variable is encountered by reducing  $x_i$ . In (TR-VAR), each type binding  $y : \kappa$  in  $\mathcal{K}$  gets (implicitly) translated to  $(y_0, \dots, y_{\ell+1}) : \kappa^\ddagger$ . In the output of translation, the last  $k+1$  components (which represent the words generated until  $x_1, \dots, x_k$  and an unknown target are encountered) are set to  $y_{\ell+1}$ , because  $x_i$ 's are unknown for the environment. In Example 8 below, we explain why we use  $y_{\ell+1}$  (rather than



$$\begin{array}{c}
\frac{}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} \Omega : \circ \rightsquigarrow (\Omega^{k+2})} \quad (\text{TR-OMEGA}) \\
\frac{}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} x_i : \circ \rightsquigarrow (\Omega^i, \mathbf{e}, \Omega^{k-i+1})} \quad (\text{TR-VARG}) \\
\frac{\mathcal{K}(y) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} y : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (y_0, y_1, \dots, y_\ell, (y_{\ell+1})^{k+1})} \quad (\text{TR-VAR}) \\
\frac{\mathcal{N}(F) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} F : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (F_0, F_1, \dots, F_\ell, (F_0)^{k+1})} \quad (\text{TR-NT}) \\
\frac{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} t : \circ \rightsquigarrow (t_0, \dots, t_{k+1})}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} a(t) : \circ \rightsquigarrow (\mathbf{br} \ a \ t_0, \dots, \mathbf{br} \ a \ t_{k+1})} \quad (\text{TR-CONST}) \\
\frac{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s : \kappa_0 \rightarrow \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{\ell+k+1}) \quad \mathbf{gar}(\kappa_0) = \ell'}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} t : \kappa_0 \rightsquigarrow (t_0, \dots, t_{\ell+k+1})} \quad (\text{TR-APP}) \\
\frac{}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s t : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (s_0(t_0, \dots, t_{\ell'}, t_{\ell'+k+1}), s_1(t_1, \dots, t_{\ell'}, t_{\ell'+k+1}), \dots, s_\ell(t_1, \dots, t_{\ell'}, t_{\ell'+k+1}), s_{\ell+1}(t_{\ell'+1}, t_1, \dots, t_{\ell'}, t_{\ell'+k+1}), \dots, s_{\ell+k+1}(t_{\ell'+k+1}, t_1, \dots, t_{\ell'}, t_{\ell'+k+1}))} \\
\frac{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s : \circ^{\ell+1} \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{\ell+k+2}) \quad \mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} t : \circ \rightsquigarrow (t_0, \dots, t_{k+1})}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s t : \circ^\ell \rightarrow \circ \rightsquigarrow (s_0 \oplus (\mathbf{br} \ s_1 \ t_0), s_2, \dots, s_{\ell+1}, s_{\ell+2} \oplus (\mathbf{br} \ s_1 \ t_1), \dots, s_{\ell+k+2} \oplus (\mathbf{br} \ s_1 \ t_{k+1}))} \quad (\text{TR-APPG}) \\
\frac{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s : \circ \rightsquigarrow (s_0, \dots, s_{k+1}) \quad \mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} t : \circ \rightsquigarrow (t_0, \dots, t_{k+1})}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} s \oplus t : \circ \rightsquigarrow (s_0 \oplus t_0, \dots, s_{k+1} \oplus t_{k+1})} \quad (\text{TR-CHOICE}) \\
\frac{\mathcal{N}(F) = \kappa_1 \rightarrow \dots \rightarrow \kappa_m \Rightarrow \circ^k \rightarrow \circ \quad y_1 : \kappa_1, \dots, y_m : \kappa_m; x_1, \dots, x_k : \circ \vdash_{\mathcal{N}} t : \circ \rightsquigarrow (t_0, t_1, \dots, t_k, t_{k+1}) \quad \tilde{y}_i = (y_{i,0}, \dots, y_{i, \mathbf{gar}(\kappa_i)+1}) \quad \tilde{y}'_i = (y_{i,1}, \dots, y_{i, \mathbf{gar}(\kappa_i)+1}) \quad (1 \leq i \leq m)}{\vdash_{\mathcal{N}} (F y_1 \dots y_m x_1 \dots x_k \rightarrow t) \rightsquigarrow \left( \begin{array}{l} \{F_0 \tilde{y}_1 \dots \tilde{y}_m \rightarrow t_0\} \cup \\ \{F_i \tilde{y}'_1 \dots \tilde{y}'_m \rightarrow t_i \mid i \in \{1, \dots, k\}\} \end{array} \right)} \quad (\text{TR-RULE}) \\
\frac{\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S \mathbf{e}) \quad \mathbf{e} \text{ does not occur in } \mathcal{R} \quad \mathcal{R}' = \bigcup \{ \mathcal{R}'' \mid \vdash_{\mathcal{N}} (F \tilde{y} \tilde{x} \rightarrow t) \rightsquigarrow \mathcal{R}'', (F \tilde{y} \tilde{x} \rightarrow t) \in \mathcal{R} \}}{\mathcal{G} \rightsquigarrow (\mathbf{br}(\Sigma), \mathcal{N}^\ddagger, \mathcal{R}', S_1)} \quad (\text{TR-GRAM})
\end{array}$$

■ **Figure 1** Translation rules from a word grammar to a tree grammar.

## 22:10 Size-Preserving Translations from Word Grammars to Tree Grammars

$y_0$ ) for the last  $k + 1$  components of (TR-VAR). There are also two rules for applications, (TR-APP) and (TR-APPG). (TR-APPG) is used when the function  $s$  has an order-1 type. The first component of the output is set to  $s_0 \oplus (\mathbf{br} s_1 t_0)$ , since  $st$  encounters the current target either when  $s$  does so without using  $t$  (which is covered by  $s_0$ ), or  $s$  uses the argument  $t$  and then  $t$  encounters the target (which is covered by  $\mathbf{br} s_1 t_0$ ). In (TR-APP), the output is just an application, but we need to choose the function and the argument appropriately for each component of the output tuple. Note that  $\kappa_0$  in (TR-APP) can be  $\mathbf{o}$ .

► **Example 8.** The following example of the translation shows why we need to use  $y_{\ell+1}$  rather than  $y_0$  for the last  $k + 1$  components in (TR-VAR). Let us consider the following grammar:

$$\begin{array}{lll} Sx \rightarrow Fx(Gx) & Fyg \rightarrow T(Ag)y & Thy \rightarrow h(hy) \\ Agy \rightarrow g(Jy) & Gxh \rightarrow hx & Jyz \rightarrow (\mathbf{b}y) \oplus (\mathbf{c}z) \\ S : \mathbf{o} \rightarrow \mathbf{o}, & F : \mathbf{o} \rightarrow ((\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o}) \rightarrow \mathbf{o}, & T : (\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o} \rightarrow \mathbf{o}, \\ A : ((\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o}) \rightarrow \mathbf{o} \rightarrow \mathbf{o}, & G : \mathbf{o} \rightarrow (\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o}, & J : \mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}. \end{array}$$

Then we have the following reduction sequence:

$$\begin{aligned} Se &\rightarrow Fe(Ge) \rightarrow T(A(Ge))e \rightarrow A(Ge)(A(Ge)e) \rightarrow Ge(J(A(Ge)e)) \\ &\rightarrow J(A(Ge)e)e \rightarrow (\mathbf{b}(A(Ge)e)) \oplus (\mathbf{c}e) \rightarrow (\mathbf{b}(Ge(Je))) \oplus (\mathbf{c}e) \\ &\rightarrow (\mathbf{b}(Jee)) \oplus (\mathbf{c}e) \rightarrow (\mathbf{b}((\mathbf{b}e) \oplus (\mathbf{c}e))) \oplus (\mathbf{c}e) \rightarrow ((\mathbf{b}(\mathbf{b}e)) \oplus (\mathbf{b}(\mathbf{c}e))) \oplus (\mathbf{c}e). \end{aligned}$$

Thus, the language generated by the grammar is  $\{\mathbf{bb}, \mathbf{bc}, \mathbf{c}\}$ . The translation produces:

$$\begin{aligned} S_1 &\rightarrow F_0(\mathbf{e}, \Omega)(G_0(\mathbf{e}, \Omega), G_0(\Omega, \Omega)) \\ F_0(y_0, y_1)(g_0, g_1) &\rightarrow (T_0(A_0(g_0, g_1), A_1(g_1), A_0(g_1, g_1))) \\ &\quad \oplus (\mathbf{br}(T_1(\quad A_1(g_1), A_0(g_1, g_1)))y_0)) \\ T_0(h_0, h_1, h_2) &\rightarrow h_0 \oplus (\mathbf{br} h_1 h_0) & A_0(g_0, g_1) &\rightarrow g_0(\mathbf{br} J_1 \Omega, J_2, \mathbf{br} J_1 \Omega) \\ T_1(\quad h_1, h_2) &\rightarrow \mathbf{br} h_1 (\mathbf{br} h_1 \mathbf{e}) & A_1(\quad g_1) &\rightarrow g_1(\mathbf{br} J_1 \mathbf{e}, J_2, \mathbf{br} J_1 \Omega) \\ G_0(x_0, x_1)(h_0, h_1, h_2) &\rightarrow h_0 \oplus (\mathbf{br} h_1 x_0) \\ J_0 &\rightarrow \Omega & J_1 &\rightarrow \mathbf{b} & J_2 &\rightarrow \mathbf{c}. \end{aligned}$$

The frontier language generated by the output grammar is  $\{\mathbf{bb}, \mathbf{bc}, \mathbf{c}\}$ . If we changed the definition of (TR-VAR) by replacing  $y_{k+1}$  with  $y_0$ , then  $A_1(g_1)$  in the body of  $F_0$  above would be replaced with  $A_1(g_0)$ . – But then we would wrongly obtain  $\mathbf{cc}$  as a member of the frontier language.

The following theorem guarantees that the output of the translation is a valid grammar; the preservation of the language will be proved later in Section 5.

► **Theorem 9.** *Suppose that  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S\mathbf{e})$  is an order- $(n + 1)$  word grammar, and  $\mathcal{G} \rightsquigarrow \mathcal{G}'$ . Then  $\mathcal{G}'$  is a (well-typed) order- $n$  tree grammar.*

The well-typedness of  $\mathcal{G}'$  follows immediately from the following lemma, which can be proved by a straightforward induction.

► **Lemma 10.** *If  $\mathcal{K}; \tilde{x} : \mathbf{o} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (\tilde{t})$ , then  $\mathcal{N}^{\ddagger}, \mathcal{K}^{\dagger} \vdash (\tilde{t}) : \kappa^{\dagger+|\tilde{x}|}$ . Furthermore for  $y \in \text{dom}(\mathcal{K})$ ,  $y_0$  does not occur freely in  $\tilde{t}$  except for  $t_0$ .*

## 4 Complexity and Application

Here we give some complexity results of our translation: an upper bound of the size of an output grammar and the time complexity of the translation. Based on the complexity results, we discuss an application of our translation.

For a grammar  $\mathcal{G}$ , we write  $A_{\mathcal{G}}$  (or  $A$  if  $\mathcal{G}$  is clear from the context) for the largest arity of the terminals, variables, and nonterminals occurring in  $\mathcal{G}$ . Note that, for an input word grammar  $\mathcal{G}$ , we have  $A_{\mathcal{G}} \leq |\mathcal{N}| \leq |\mathcal{G}|_{\text{ch}}$  and  $|\mathcal{G}|_{\text{cy}} \leq |\mathcal{G}|_{\text{ch}}$ , but it is not necessary the case that  $A_{\mathcal{G}} \leq |\mathcal{G}|_{\text{cy}}$  (for any  $\kappa$ ,  $\{S \rightarrow F(G\mathbf{e}), F f^{\kappa \rightarrow \circ} \rightarrow \mathbf{e}, G x^{\circ} g^{\kappa} \rightarrow \mathbf{e}\}$  is well-typed and  $A_{\mathcal{G}} \geq \text{ari}(\kappa)$ ). Also note that  $A_{\mathcal{G}} \geq 1$  for an input grammar  $\mathcal{G}$  (since the start term is  $S\mathbf{e}$ ). For  $r = (F \tilde{x} \rightarrow t) \in \mathcal{R}$ , we define  $|r| := |t| + \text{ari}(\mathcal{N}(F))$ , so that  $|\mathcal{G}|_{\text{cy}} = |t^{\circ}| + \sum_{r \in \mathcal{R}} |r|$ ; also, we write  $\mathcal{R}_r$  for  $\mathcal{R}'$  such that  $\vdash_{\mathcal{N}} r \rightsquigarrow \mathcal{R}'$  by (TR-RULE).

### 4.1 Complexity

To suppress the blow-up of the cost of our translation, we pre-process an input grammar to convert it to a certain normal form, as in [8]. A grammar  $\mathcal{G}$  is in *normal form* if each rule is of the form

$$F \tilde{z} \rightarrow (f_{1,0}(f_{1,1}\tilde{z}_{1,1}) \cdots (f_{1,\ell_1}\tilde{z}_{1,\ell_1})) \oplus \cdots \oplus (f_{k,0}(f_{k,1}\tilde{z}_{k,1}) \cdots (f_{k,\ell_k}\tilde{z}_{k,\ell_k})),$$

where each  $f_{h,i}$  is a non-terminal, terminal, or variable; and each  $z_{h,i,j}$  is a variable; note that  $k$  and each  $\ell_h$  can be 0. (Rigorously, the expression like  $t_1 \oplus \cdots \oplus t_k$  can be represented as  $D[t_1] \cdots [t_k]$  with  $D ::= [] \mid \Omega \mid D \oplus D$ .) By the following transformation, any grammar  $\mathcal{G}$  can be transformed to a grammar in normal form: First note that any rule is of the form

$$F \tilde{z} \rightarrow (f_{1,0} t_{1,1} \cdots t_{1,\ell_1}) \oplus \cdots \oplus (f_{k,0} t_{k,1} \cdots t_{k,\ell_k})$$

where each  $f_{h,i}$  is a non-terminal, terminal, or variable, and each  $t_{h,i}$  is a term (which may contain  $\oplus$  and  $\Omega$ ). If there exist  $h$  and  $i$  such that  $t_{h,i}$  is not of the form  $f_{h,i}\tilde{z}_{h,i}$ , then we replace the occurrence  $t_{h,i}$  with  $G\tilde{z}_{h,i}$  and add the rule  $G\tilde{z}_{h,i}\tilde{y} \rightarrow t_{h,i}\tilde{y}$ , where  $G$  is a fresh non-terminal and  $\{\tilde{z}_{h,i}\} (\subseteq \{\tilde{z}\})$  is all the free variables occurring in  $t_{h,i}$ . By repeated applications of this transformation, we obtain a grammar in normal form; note that the number of repeated applications is at most  $|\mathcal{G}|_{\text{cy}}$ . The following lemma guarantees that the preprocessing transformation does not blow up the grammar size; see the full version for the proof.

► **Lemma 11.** *Let  $\mathcal{G}' = (\Sigma, \mathcal{N}', \mathcal{R}', S\mathbf{e})$  be the grammar obtained from  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S\mathbf{e})$  by the above transformation. We write  $A$  for  $A_{\mathcal{G}}$ . Then,*

1.  $A_{\mathcal{G}'} \leq 2A$ ,
2.  $|\mathcal{N}'| \leq O(|\mathcal{N}| \times |\mathcal{G}|_{\text{cy}})$
3.  $|\mathcal{G}'|_{\text{cy}} \in O(A \times |\mathcal{G}|_{\text{cy}})$ ,
4. *the time complexity of the normal-form transformation is  $O(A \times |\mathcal{G}|_{\text{cy}})$ ,*
5.  $\text{ord}(\mathcal{G}') = \text{ord}(\mathcal{G})$ .

We obtain the following complexity results; see Appendix A for the proof.

► **Theorem 12.** *Let  $\mathcal{G}'$  be the output of our translation (i.e., the composite of the normal-form transformation and that of Fig. 1) for an input grammar  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S\mathbf{e})$ . We write  $A$  for  $A_{\mathcal{G}}$ . Then,*

1.  $A_{\mathcal{G}'} \in O(A^2)$ ,
2.  $|\mathcal{G}'|_{\text{cy}} \in O(A^4 \times |\mathcal{G}|_{\text{cy}})$  and  $|\mathcal{G}'|_{\text{ch}} \in O(A^4 \times |\mathcal{G}|_{\text{cy}} + (|\mathcal{N}| \times |\mathcal{G}|_{\text{cy}})^n)$  where  $n = \text{ord}(\mathcal{G})$ ,
3. *the time complexity of our translation is  $O((A^3 + |\mathcal{N}|) \times |\mathcal{G}|_{\text{cy}})$ .*

## 4.2 Applications

Our translation can be used to bridge results on higher-order word and tree grammars. Besides applications of our previous translation [1], our new *size-preserving* transformation can be applied in contexts where the complexity is critical. For example, Parys [12] showed that the diagonal problem for higher-order grammars (which is called *nondeterministic higher-order recursion schemes* in [12]) is  $(n - 1)$ -EXPTIME-complete for order- $n$  word grammars, and  $n$ -EXPTIME-complete for order- $n$  tree grammars. In his paper, the two results, for words and for trees, were proved separately. By using the result in the present paper, however, the  $(n - 1)$ -EXPTIME-completeness of the diagonal problem for word grammars of order  $n$  immediately follows from the  $n$ -EXPTIME-completeness of the diagonal problem for tree grammars of order  $n$  (where, on the hardness part, use the converse transformation given in [1, Theorem 5]).

## 5 Correctness of the Translation

Here we prove the following theorem, which states the correctness of the translation.

► **Theorem 13.** *If  $\mathcal{G} \rightsquigarrow \mathcal{G}'$ , then  $\mathcal{L}_w(\mathcal{G}) = \mathcal{L}_{\text{leaf}}(\mathcal{G}') \uparrow_e$ .*

We show this theorem in the following steps:

- First, in Section 5.1, we reduce the proof to the case where  $\mathcal{G}$  is *recursion-free* (Lemma 15). This is rather a standard technique, which uses the *finite approximation*  $\mathcal{G}^{(m)}$ : the reduction of  $\mathcal{G}^{(m)}$  imitates that of  $\mathcal{G}$  up to  $m$ -steps, but diverges after  $m$ -steps.
- Then we show the statement of Theorem 13 with the assumption that  $\mathcal{G}$  is recursion-free (Lemma 21). This is the main part, proved by using the subject reduction property. In the proof of the subject reduction, we modify the reduction  $\rightarrow_{\mathcal{G}}$  of the source grammar  $\mathcal{G}$ , by using *explicit substitutions*. We define the modified reduction in Section 5.2, and then we show the subject reduction and the correctness for a recursion-free grammar in Section 5.3.

### 5.1 Reduction of the correctness to recursion-free grammars

► **Definition 14** (recursion-free grammars). *A grammar  $\mathcal{G}$  is called recursion-free if there is no cyclic dependency on its non-terminals. Precisely, we define a binary relation  $\succ_{\mathcal{G}}$  on nonterminals of  $\mathcal{G}$  by:  $F \succ_{\mathcal{G}} F'$  iff  $F'$  occurs on the right-hand side of the rule for  $F$ ; then  $\mathcal{G}$  is recursion free if the transitive closure  $\succ_{\mathcal{G}}^*$  of  $\succ_{\mathcal{G}}$  is irreflexive (i.e.,  $F \succ_{\mathcal{G}}^* F$  for no  $F \in \mathcal{N}$ ).*

Here we show the following lemma:

► **Lemma 15.** *Suppose that, for any  $\mathcal{G}$  and  $\mathcal{G}'$ , if  $\mathcal{G}$  is recursion-free and  $\mathcal{G} \rightsquigarrow \mathcal{G}'$ , then  $\mathcal{L}_w(\mathcal{G}) = \mathcal{L}_{\text{leaf}}(\mathcal{G}') \uparrow_e$ . Then, for any  $\mathcal{G}$  and  $\mathcal{G}'$ , if  $\mathcal{G} \rightsquigarrow \mathcal{G}'$ , then  $\mathcal{L}_w(\mathcal{G}) = \mathcal{L}_{\text{leaf}}(\mathcal{G}') \uparrow_e$ .*

To show this lemma, we define the finite approximation mentioned above. Given terms  $t$ ,  $s_i$  ( $i \in I$ ) and variables  $x_i$  ( $i \in I$ ), we write  $[s_i/x_i]_{i \in I} t$  for the term obtained by simultaneously substituting  $s_i$  for  $x_i$  in  $t$ . Given  $m \geq 0$  and  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, t^\circ)$ , we define the  $m$ -th approximation  $\mathcal{G}^{(m)}$  of  $\mathcal{G}$  as follows.

$$\begin{aligned} \mathcal{N}^{(m)} &:= \{F^{(i)} \mapsto \mathcal{N}(F) \mid F \in \text{dom}(\mathcal{N}), 0 \leq i \leq m\} \\ t_{\mathcal{N}}^{(i)} &:= [F^{(i)}/F]_{F \in \mathcal{N}} t \quad (\text{for any term } t \text{ and } i \in \{0, \dots, m\}) \\ \mathcal{R}_{\mathcal{N}}^{(m)} &:= \{F^{(i)} \tilde{x} \rightarrow t_{\mathcal{N}}^{(i-1)} \mid (F \tilde{x} \rightarrow t) \in \mathcal{R}, 1 \leq i \leq m\} \\ &\quad \cup \{F^{(0)} \tilde{x} \rightarrow \Omega \mid (F \tilde{x} \rightarrow t) \in \mathcal{R}\} \\ \mathcal{G}^{(m)} &:= (\Sigma, \mathcal{N}^{(m)}, \mathcal{R}_{\mathcal{N}}^{(m)}, (t^\circ)_{\mathcal{N}}^{(m)}). \end{aligned}$$

We write  $t^{(i)}$  and  $\mathcal{R}^{(m)}$  for  $t_{\mathcal{N}}^{(i)}$  and  $\mathcal{R}_{\mathcal{N}}^{(m)}$  if  $\mathcal{N}$  is clear from the context.

The following are basic properties of  $\mathcal{G}^{(m)}$ :

► **Lemma 16.**

1. If  $\mathcal{G}$  is in the domain of the transformation  $\rightsquigarrow$  (i.e.,  $\mathcal{G}$  is of the form  $(\Sigma, \mathcal{N}, \mathcal{R}, S \mathbf{e})$  where  $\Sigma$  is a word alphabet and  $\mathcal{R}$  does not contain  $\mathbf{e}$ ), then so is  $\mathcal{G}^{(m)}$ .
2.  $\mathcal{G}^{(m)}$  is a recursion-free grammar.
3.  $\mathcal{L}(\mathcal{G}) = \cup_m \mathcal{L}(\mathcal{G}^{(m)})$ . Hence,  $\mathcal{L}_{\mathbf{w}}(\mathcal{G}) = \cup_m \mathcal{L}_{\mathbf{w}}(\mathcal{G}^{(m)})$  for any word grammar  $\mathcal{G}$ , and  $\mathcal{L}_{\text{leaf}}(\mathcal{G})\uparrow_{\mathbf{e}} = \cup_m \mathcal{L}_{\text{leaf}}(\mathcal{G}^{(m)})\uparrow_{\mathbf{e}}$  for any br-grammar  $\mathcal{G}$ .

**Proof.** Item 1 is clear, and also Item 2 is clear (since  $F^{(i)} \succ_{\mathcal{G}^{(m)}} G^{(j)}$  implies  $j = i - 1$ ).

Item 3: Let  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, t^\circ)$ . To show  $\cup_m \mathcal{L}(\mathcal{G}^{(m)}) \subseteq \mathcal{L}(\mathcal{G})$ , let  $v \in \mathcal{L}(\mathcal{G}^{(m)})$ ; i.e., there exist a choice context  $C$  and a reduction sequence  $\Pi$  in  $\mathcal{G}^{(m)}$  from  $(t^\circ)^{(m)}$  to  $C[v]$ . Let us call a rule in  $\{F^{(0)} \tilde{x} \rightarrow \Omega \mid (F \tilde{x} \rightarrow t) \in \mathcal{R}\}$  a *bottom rule*. We can easily show the following fact: if  $t \rightarrow s$  by some bottom rule  $r$  and  $s \rightarrow u$ , then there is a term  $u'$  such that  $t \rightarrow u'$  and  $u' \rightarrow^* u$  where the latter rewriting uses only  $r$ . By using this fact repeatedly, from  $\Pi$ , we obtain a choice context  $C'$  and a reduction sequence  $\Pi'$  in  $\mathcal{G}^{(m)}$  from  $(t^\circ)^{(m)}$  to  $C'[v]$  where  $\Pi'$  does not use any bottom rule. Then, by dropping the index  $i$  of every nonterminal  $F^{(i)}$  in  $\Pi'$ , we obtain a reduction sequence in  $\mathcal{G}$  from  $t^\circ$  to  $C''[v]$  for some  $C''$ .

Conversely, let  $v \in \mathcal{L}(\mathcal{G})$ . We have  $C$  and a reduction sequence  $\Pi$  in  $\mathcal{G}$  from  $t^\circ$  to  $C[v]$ ; let  $m$  be the length of  $\Pi$ . Then there exist  $C'$  and a reduction sequence  $\Pi'$  in  $\mathcal{G}^{(m)}$  from  $(t^\circ)^{(m)}$  to  $C'[v]$  (such that  $\Pi$  is obtained from  $\Pi'$  by dropping the indices of all the nonterminals in  $\Pi'$ ). Hence  $v \in \mathcal{L}(\mathcal{G}^{(m)})$ , and thus  $\mathcal{L}(\mathcal{G}) \subseteq \cup_m \mathcal{L}(\mathcal{G}^{(m)})$ . ◀

The following lemma is trivial; see the full version for the proof.

► **Lemma 17.** If  $\mathcal{G} \rightsquigarrow \mathcal{G}'$  and  $\mathcal{G}^{(m)} \rightsquigarrow \mathcal{G}'_m$ , then  $\mathcal{L}_{\text{leaf}}(\mathcal{G}^{(m)})\uparrow_{\mathbf{e}} = \mathcal{L}_{\text{leaf}}(\mathcal{G}'_m)\uparrow_{\mathbf{e}}$ .

Now we can show:

**Proof of Lemma 15.** Let  $\mathcal{G} \rightsquigarrow \mathcal{G}'$  and  $\mathcal{G}^{(m)} \rightsquigarrow \mathcal{G}'_m$  for each  $m$ . Then, by the assumption,

$$\mathcal{L}_{\mathbf{w}}(\mathcal{G}^{(m)}) = \mathcal{L}_{\text{leaf}}(\mathcal{G}'_m)\uparrow_{\mathbf{e}}. \quad (1)$$

Then,

$$\begin{aligned} \mathcal{L}_{\mathbf{w}}(\mathcal{G}) &= \cup_m \mathcal{L}_{\mathbf{w}}(\mathcal{G}^{(m)}) && \text{(by Lemma 16-3)} \\ &= \cup_m \mathcal{L}_{\text{leaf}}(\mathcal{G}'_m)\uparrow_{\mathbf{e}} && \text{(by (1))} \\ &= \cup_m \mathcal{L}_{\text{leaf}}(\mathcal{G}'^{(m)})\uparrow_{\mathbf{e}} && \text{(by Lemma 17)} \\ &= \mathcal{L}_{\text{leaf}}(\mathcal{G}')\uparrow_{\mathbf{e}}. && \text{(by Lemma 16-3).} \end{aligned} \quad \blacktriangleleft$$

## 5.2 The modified reduction of source grammars

As explained at the beginning of this section, we modify the reduction relation of a word grammar  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, t^\circ)$  by using explicit substitutions. We first extend the set of terms as follows, which we call *extended terms*:

$$t ::= x \mid \mathbf{e} \mid a(t) \mid t_1 t_2 \mid t_1 \oplus t_2 \mid \Omega \mid \{t_1/x_1, \dots, t_\ell/x_\ell\}t_0$$

Here, we write  $a(t)$  instead of  $at$ , to emphasize that  $a$  is a unary constructor. Recall that nonterminals are included in variables  $x$ . The term  $\{t_1/x_1, \dots, t_\ell/x_\ell\}t_0$  is an *explicit substitution* and is limited to the ground type:

$$\frac{\mathcal{K} \vdash_{\text{ST}} t_i : \circ \quad (i \in \{1, \dots, \ell\}) \quad \mathcal{K}, x_1 : \circ, \dots, x_\ell : \circ \vdash_{\text{ST}} t_0 : \circ}{\mathcal{K} \vdash_{\text{ST}} \{t_1/x_1, \dots, t_\ell/x_\ell\}t_0 : \circ}$$

We call the original notions of a term, evaluation context, and choice context a *non-extended term*, *non-extended evaluation context*, and *non-extended choice context*, respectively, if we need the clarity. We use the same meta-variables  $s, t, u$  for the extended terms, and also use  $E$  and  $C$  for the extended evaluation contexts and extended choice contexts defined below. We avoid this ambiguity as follows: in this subsection, these meta-variables range over the extended notions unless we declare otherwise; in the next subsection (i.e., in Section 5.3 for the proof of the correctness for recursion-free grammars), the source term of the transformation relation  $\rightsquigarrow$  is an extended one and the target term is a non-extended one.

We define the *extended evaluation contexts* by the following grammar:

$$E ::= [] \mid a(E) \mid E \oplus t \mid t \oplus E \mid \{t_1/x_1, \dots, t_\ell/x_\ell\}E.$$

We often abbreviate  $\{t_1/x_1, \dots, t_\ell/x_\ell\}t_0$  and  $\{t_1/x_1, \dots, t_\ell/x_\ell\}E$  to  $\{\tilde{t}/\tilde{x}\}t_0$  and  $\{\tilde{t}/\tilde{x}\}E$ , respectively. Then the *modified reduction*, written as  $\rightarrow_{\text{es}, \mathcal{G}}$  (or  $\rightarrow_{\text{es}}$  if  $\mathcal{G}$  is clear), is defined as follows:

$$\frac{\mathcal{N}(F) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \quad F \tilde{y} \tilde{z} \rightarrow u \in \mathcal{R} \quad |\tilde{\kappa}| = |\tilde{y}| = |\tilde{s}| \quad \ell = |\tilde{z}| = |\tilde{t}| \quad \tilde{z} \text{ do not occur in } E[F \tilde{s} \tilde{t}]}{E[F \tilde{s} \tilde{t}] \rightarrow_{\text{es}} E[\{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u]}$$

$$\frac{}{E[\{\tilde{s}/\tilde{z}\}z_i] \rightarrow_{\text{es}} E[s_i]} \quad \frac{x \notin \{z_1, \dots, z_{|\tilde{s}|}\} \cup \text{dom}(\mathcal{N})}{E[\{\tilde{s}/\tilde{z}\}x] \rightarrow_{\text{es}} E[x]}$$

$$\frac{}{E[\{\tilde{s}/\tilde{z}\}e] \rightarrow_{\text{es}} E[e]} \quad \frac{}{E[\{\tilde{s}/\tilde{z}\}a(t)] \rightarrow_{\text{es}} E[a(\{\tilde{s}/\tilde{z}\}t)]}$$

$$\frac{}{E[\{\tilde{s}/\tilde{z}\}(t_1 \oplus t_2)] \rightarrow_{\text{es}} E[(\{\tilde{s}/\tilde{z}\}t_1) \oplus (\{\tilde{s}/\tilde{z}\}t_2)]} \quad \frac{}{E[a(t_1 \oplus t_2)] \rightarrow_{\text{es}} E[a(t_1) \oplus a(t_2)]}$$

We define the *extended choice contexts* by:  $C ::= [] \mid C \oplus t \mid t \oplus C$ . For  $\mathcal{N} \vdash_{\text{ST}} t : \circ$ , we define the languages generated by a term and by a grammar with respect to  $\rightarrow_{\text{es}}$  as follows:

$$\mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}, t) = \{\mathbf{word}(v) \mid t \rightarrow_{\text{es}}^* C[v]\} \quad \mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}) = \mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}, t^\circ).$$

There is an obvious function  $\psi$  from the extended terms to the non-extended terms that performs every explicit substitution as the real substitution. The following lemma can be proved in a standard manner; see the full version for the proof.

► **Lemma 18.** *Let  $\mathcal{G}$  be a word grammar.*

1. *For any extended term  $\mathcal{N} \vdash_{\text{ST}} t : \circ$ , we have  $\mathcal{L}_{\mathbf{w}}(\mathcal{G}, \psi(t)) = \mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}, t)$ . Especially,  $\mathcal{L}_{\mathbf{w}}(\mathcal{G}) = \mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G})$ .*
2. *For any extended terms  $\mathcal{N} \vdash t, t' : \circ$ , if  $t \rightarrow_{\text{es}} t'$ , then  $\mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}, t) = \mathcal{L}_{\mathbf{w}}^{\text{es}}(\mathcal{G}, t')$ .*

We also extend the transformation relation to handle explicit substitutions:

$$\frac{\mathcal{K}; x_1, \dots, x_k : \circ \vdash s_i : \circ \rightsquigarrow (s_{i,0}, \dots, s_{i,k+1}) \quad (i \in \{1, \dots, \ell\}) \quad \mathcal{K}; z_1, \dots, z_\ell, x_1, \dots, x_k : \circ \vdash t : \circ \rightsquigarrow (t_0, \dots, t_{\ell+k+1})}{\mathcal{K}; x_1, \dots, x_k : \circ \vdash \{\tilde{s}/\tilde{z}\}t : \circ \rightsquigarrow (t_0 \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} t_i s_{i,0}), \dots, t_{\ell+k+1} \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} t_i s_{i,k+1}))} \quad (\text{TR-SUB})$$

### 5.3 Correctness for recursion-free grammars

To show the correctness for recursion-free grammars, we use the following substitution lemma and the subject reduction property:

► **Lemma 19** (substitution lemma). *Suppose that  $u$  contains no explicit substitutions (i.e., no subterms of the form  $\{\tilde{t}'/\tilde{z}'\}s'$ ) and*

$$\begin{aligned} \tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash u : \kappa' &\rightsquigarrow (u_0, \dots, u_{\mathbf{gar}(\kappa') + |\tilde{z}|}, u_{\mathbf{gar}(\kappa') + |\tilde{z}| + 1}) \\ \tilde{x} : \circ \vdash s'_i : \kappa_i &\rightsquigarrow (s'_{i,0}, \dots, s'_{i,\ell_i + k + 1}) \quad \ell_i = \mathbf{gar}(\kappa_i) \quad (i = 1, \dots, |\tilde{\kappa}|) \\ k = |\tilde{x}| \quad \{\tilde{x}\} \cap \{\tilde{z}\} &= \emptyset. \end{aligned}$$

We define the following substitution functions

$$\begin{aligned} \theta_j &= \theta_{1,j} \cdots \theta_{|\tilde{\kappa}|,j} \quad (j = 0, \dots, k) \\ \theta_{i,0} &= [s'_{i,0}/y_{i,0}, s'_{i,1}/y_{i,1}, \dots, s'_{i,\ell_i}/y_{i,\ell_i}, s'_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \quad (i = 1, \dots, |\tilde{\kappa}|) \\ \theta_{i,j} &= [s'_{i,\ell_i+j}/y_{i,0}, s'_{i,1}/y_{i,1}, \dots, s'_{i,\ell_i}/y_{i,\ell_i}, s'_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \\ &\quad (i = 1, \dots, |\tilde{\kappa}|, j = 1, \dots, k + 1). \end{aligned}$$

Then we have:

1.  $\tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]u : \kappa' \rightsquigarrow (\theta_0 u_0, \dots, \theta_0 u_{\mathbf{gar}(\kappa') + |\tilde{z}|}, \theta_1 u_0, \dots, \theta_k u_0, \theta_0 u_{\mathbf{gar}(\kappa') + |\tilde{z}| + 1})$
2.  $\theta_0 u_{\mathbf{gar}(\kappa') + |\tilde{z}| + 1} = \theta_{k+1} u_0$ .

**Proof.** Both items can be shown by induction on  $u$  and case analysis on the last rule used for the derivation  $\tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash u : \kappa' \rightsquigarrow (u_0, \dots, u_{\mathbf{gar}(\kappa') + |\tilde{z}|})$ . See Appendix B. ◀

► **Lemma 20** (subject reduction). *If  $x_1, \dots, x_k : \circ \vdash s : \circ \rightsquigarrow (s_0, \dots, s_{k+1})$  and  $s \rightarrow_{\text{es}} t$ , then there exist  $t_0, \dots, t_{k+1}$  such that  $x_1, \dots, x_k : \circ \vdash t : \circ \rightsquigarrow (t_0, \dots, t_{k+1})$  and  $s_i \approx t_i$  for each  $i \in \{1, \dots, k + 1\}$ .*

**Proof.** We use Lemma 19. The proof proceeds by induction on the derivation of  $\tilde{x} : \circ \vdash s : \circ \rightsquigarrow (s_0, \dots, s_{k+1})$  and case analysis on evaluation contexts and redexes. See Appendix B. ◀

Now we show the correctness for recursion-free grammars; as explained already, Theorem 13 follows immediately from this and Lemma 15.

► **Lemma 21.** *Suppose that  $\mathcal{G}$  is recursion-free. If  $\mathcal{G} \rightsquigarrow \mathcal{G}'$ , then  $\mathcal{L}_w(\mathcal{G}) = \mathcal{L}_{\text{leaf}}(\mathcal{G}') \uparrow_e$ .*

**Proof.** Since  $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S\mathbf{e})$  is recursion-free, every term  $x : \circ \vdash s : \circ$  is strongly normalizing with respect to  $\rightarrow_{\text{es}}$ . Since the reduction relation is finitely branching, the length of reduction sequences from  $s$  is bounded. Let  $\#(s)$  be the length of the longest reduction sequence from  $s$ .

Now we show that

$$x : \circ \vdash_{\mathcal{N}} s : \circ \rightsquigarrow (s_0, s_1, s_2) \quad \text{implies} \quad \mathcal{L}_w^{\text{es}}(\mathcal{G}, [e/x]s) = \mathcal{L}_{\text{leaf}}(\mathcal{G}', s_1) \uparrow_e \quad (2)$$

by induction on  $\#(s)$ . If (2) holds, then we can complete the proof as follows: let the rule of  $S$  be  $Sx \rightarrow s$ ; then  $\mathcal{G}'$  has the rule  $S_1 \rightarrow s_1$ , and by Lemma 18-1 and (2) we have

$$\mathcal{L}_w(\mathcal{G}) = \mathcal{L}_w^{\text{es}}(\mathcal{G}, S\mathbf{e}) = \mathcal{L}_w^{\text{es}}(\mathcal{G}, [e/x]s) = \mathcal{L}_{\text{leaf}}(\mathcal{G}', s_1) \uparrow_e = \mathcal{L}_{\text{leaf}}(\mathcal{G}') \uparrow_e.$$

In the base case that  $\#(s) = 0$ , i.e., in the case where  $s$  is a normal form, first note that, given a term  $t$  satisfies  $x_1 : \circ, \dots, x_n : \circ \vdash_{\text{ST}} t : \circ$  for some  $x_1, \dots, x_n$ , then  $t$  is a normal form with respect to  $\rightarrow_{\text{es}}$  iff  $t$  is generated by the following grammar with start symbol  $w$ :

$$w ::= r \mid w \oplus w \quad r ::= \delta \mid x \mid a(r) \quad (a \in \Sigma) \quad \delta ::= \Omega \mid \{t_1/x_1, \dots, t_k/x_k\}\delta.$$

Then we can show (2) by induction on this grammar. (For the case  $s = \delta$ , show that  $x_1, \dots, x_k : \circ \vdash \delta : \circ \rightsquigarrow (\bar{s})$  implies  $s_i \approx \Omega$  for every  $i$ , by induction on  $\delta$ .)

In the case  $\#(s) > 0$ , suppose  $s \rightarrow_{\text{es}} s'$ . Hence  $[e/x]s \rightarrow_{\text{es}} [e/x]s'$ . By Lemma 20, there exist  $s'_0, s'_1$ , and  $s'_2$  such that  $x : \circ \vdash_{\mathcal{N}} s' : \circ \rightsquigarrow (s'_0, s'_1, s'_2)$  and  $s_1 \approx s'_1$ . By the induction hypothesis,  $\mathcal{L}_w^{\text{es}}(\mathcal{G}, [e/x]s') = \mathcal{L}_{\text{leaf}}(\mathcal{G}', s'_1)\uparrow_e$ . Then, by Lemma 18-2 and  $s'_1 \approx s_1$ , we have

$$\mathcal{L}_w^{\text{es}}(\mathcal{G}, [e/x]s) = \mathcal{L}_w^{\text{es}}(\mathcal{G}, [e/x]s') = \mathcal{L}_{\text{leaf}}(\mathcal{G}', s'_1)\uparrow_e = \mathcal{L}_{\text{leaf}}(\mathcal{G}', s_1)\uparrow_e. \quad \blacktriangleleft$$

## 6 Conclusion

We have given a new transformation that converts an order- $(n+1)$  word grammar to an order- $n$  tree grammar whose frontier language coincides with the word language of the input grammar, and have proved that, unlike the previous transformations [1, 12], our transformation is size-preserving, in that the size of the output grammar is polynomial in the size of an input grammar. The time complexity is also polynomial in the size of input grammar. These properties allow us to establish a link between algorithms for higher-order word and tree grammars. As a concrete example of this, we have also applied our result to the work of Parys [12] on the complexity of the diagonal problems for word and tree languages..

---

### References

- 1 Kazuyuki Asada and Naoki Kobayashi. On word and frontier languages of unsafe higher-order grammars. *CoRR*, abs/1604.01595, 2016. A summary has been published in Proceedings of ICALP 2016. URL: <http://arxiv.org/abs/1604.01595>.
- 2 Kazuyuki Asada and Naoki Kobayashi. Pumping lemma for higher-order languages. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 97:1–97:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.97.
- 3 Kazuyuki Asada and Naoki Kobayashi. Lambda-definable order-3 tree functions are well-quasi-ordered. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, volume 122 of *LIPIcs*, pages 14:1–14:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.14.
- 4 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009. doi:10.2168/LMCS-5(1:3)2009.
- 5 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.
- 6 Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20(2):95–207, 1982. doi:10.1016/0304-3975(82)90009-3.
- 7 Teodor Knapik, Damian Niwiński, Paweł Urzyczyn, and Igor Walukiewicz. Unsafe grammars and panic automata. In *32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*, volume 3580 of *LNCS*, pages 1450–1461. Springer, 2005. doi:10.1007/11523468\_117.
- 8 Naoki Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013. doi:10.1145/2487241.2487246.
- 9 Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. On the termination problem for probabilistic higher-order recursive programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*, pages 1–14. IEEE, 2019. doi:10.1109/LICS.2019.8785679.
- 10 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011. doi:10.2168/LMCS-7(4:9)2011.



- 11 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*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.
- 12 Paweł Parys. The Complexity of the Diagonal Problem for Recursion Schemes. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2017.45.
- 13 Mitchell Wand. An algebraic formulation of the Chomsky hierarchy. In *Category Theory Applied to Computation and Control*, volume 25 of *LNCS*, pages 209–213. Springer, 1974. doi:10.1007/3-540-07142-3\_84.

## A Proof of Theorem 12

**Proof of Theorem 12.** On Item 1, see the full version.

Next, we show Item 2: note that, by Lemma 11, we only need to show the following: if  $\mathcal{G}$  is already in normal form, then  $|\mathcal{G}'|_{\text{cy}} \in O(A^3 \times |\mathcal{G}|_{\text{cy}})$  and  $|\mathcal{G}'|_{\text{ch}} \in O(A^3 \times |\mathcal{G}|_{\text{cy}} + |\mathcal{N}|^n)$ .

Let:  $r = (F \tilde{y} \tilde{x} \rightarrow s) \in \mathcal{R}$ ,  $\mathcal{N}(F) = \kappa'_1 \rightarrow \cdots \rightarrow \kappa'_m \Rightarrow \mathfrak{o}^k \rightarrow \mathfrak{o}$ ,  $\mathcal{K} = y_1 : \kappa'_1, \dots, y_m : \kappa'_m$ , and  $\mathfrak{D}_0$  be the derivation tree of  $\mathcal{K}$ ;  $\tilde{x} : \mathfrak{o} \vdash_{\mathcal{N}} s : \mathfrak{o} \rightsquigarrow (s_0, \dots, s_{k+1})$ .

Further, let  $\mathcal{K}; \tilde{x} : \mathfrak{o} \vdash_{\mathcal{N}} t : \tilde{\kappa} \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o} \rightsquigarrow (\tilde{t})$  be an arbitrary judgment in  $\mathfrak{D}_0$ , and  $\mathfrak{D}$  be the sub-derivation-tree of  $\mathfrak{D}_0$  for this judgment. Then we can show  $\ell \leq A$  by induction on the height of  $\mathfrak{D}$ . Next, let  $h$  ( $h'$ , resp.) be the largest number, in the depth direction, of uses of (TR-APP) ((TR-APPG), resp.) in  $\mathfrak{D}$ : precisely,  $h$  ( $h'$ , resp.) is defined as the least number  $p$  such that in any path of  $\mathfrak{D}$ , the number of (TR-APP) ((TR-APPG), resp.) in the path is no more than  $p$ . Then for each  $i \in \{0, \dots, \ell + k + 1\}$ , we can show

$$|t_i| \leq 2(A + 2)^h 2^{h'} |t| \quad (3)$$

by induction on  $t$  and case analysis on the last rule of  $\mathfrak{D}$ .

Now let us consider the case  $\mathfrak{D} = \mathfrak{D}_0$  (and hence  $t = s$ ). Since  $\mathcal{G}$  is in normal form,  $h, h' \leq 2$ ; hence by (3),  $|s_i| \leq O(A^2 \times |s|)$  for each  $i$ . Also note that, from Item 1,  $\text{ari}(\mathcal{N}^\dagger(F_i)) \leq O(A^2)$  for each  $i$ . Thus, for each  $r' \in \mathcal{R}_r$ , we have  $|r'| \leq O(A^2 \times |r|)$ . Since the cardinality of  $\mathcal{R}_r$  is no more than  $A + 1$ , we have:

$$|\mathcal{G}'|_{\text{cy}} = |S_1| + \sum_{r \in \mathcal{R}} \left( \sum_{r' \in \mathcal{R}_r} |r'| \right) \leq |S \mathfrak{e}| + \sum_{r \in \mathcal{R}} O(A^3 \times |r|) \leq O(A^3 \times |\mathcal{G}|_{\text{cy}}).$$

Next we calculate  $|\mathcal{G}'|_{\text{ch}}$ . We define  $|\kappa_1 \times \cdots \times \kappa_k| := |\kappa_1| + \cdots + |\kappa_k| + k - 1$ . We can show that  $|\kappa^\dagger|, |\kappa^{\dagger'}| \leq 3|\kappa|^{\text{ord}(\kappa)}$  simultaneously by induction on  $\kappa$ . Hence, we have

$$|\mathcal{N}^\dagger| \leq \sum_{F \in \mathcal{N}} |\mathcal{N}(F)^\dagger| \leq \sum_{F \in \mathcal{N}} 3|\mathcal{N}(F)|^{\text{ord}(\mathcal{N}(F))} \leq 3 \sum_{F \in \mathcal{N}} |\mathcal{N}(F)|^n \leq 3|\mathcal{N}|^n$$

and thus  $|\mathcal{G}'|_{\text{ch}} \leq O(A^3 \times |\mathcal{G}|_{\text{cy}} + |\mathcal{N}|^n)$ .

Lastly, we show Item 3. Note that, by Lemma 11, we only need to show the following: if  $\mathcal{G}$  is already in normal form, then the transformation given in Fig. 1 takes  $O(A^2|\mathcal{G}|_{\text{cy}} + |\mathcal{N}|)$  time.

We implement terms by directed acyclic graphs. The implementation  $[\kappa]$  of a type  $\kappa$  is as follows: Let  $\kappa$  be of the form  $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$ . If  $k = 0$ ,  $[\kappa]$  is the 2-length array  $[k, \ell]$ . If  $k > 0$ , note that  $\kappa = \kappa_1 \rightarrow (\kappa_2 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})$  and we define  $[\kappa]$  as the 4-length array  $[k, \ell, [\kappa_1], [\kappa_2 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}]]$ . Then, the two operations that respectively extract  $k$  and  $\ell$  from a type  $\kappa = \kappa_1 \rightarrow \cdots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$  and the type

construction  $\kappa \rightarrow \kappa'$  from  $\kappa$  and  $\kappa'$  can be performed by  $O(1)$ . The operation that extracts the sequence  $\kappa_1, \dots, \kappa_k$  from a type  $\kappa = \kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$  can be performed by  $O(\text{ari}(\kappa))$ . During the computation of  $\mathcal{G}'$  from  $\mathcal{G}$ , we need to extract such a sequence  $\kappa_1, \dots, \kappa_k$  just at the top of the derivation tree of (TR-RULE) for each rewriting rule; then in the derivation tree, all the typing environments are common and can be shared.

Now let us consider the computation of a derivation tree; look at Fig. 1. To compute the base transformation rules (i.e., (TR-OMEGA), (TR-VARG), (TR-VAR), and (TR-NT)), it takes  $O(A)$  time, as the length of the resulting tuple is bounded by  $O(A)$ . The computation for one occurrence of (TR-CHOICE) in the derivation tree takes  $O(k)$  time, i.e., if the computations for  $s$  and  $t$  take  $p$  and  $q$  time, respectively, then that for  $s \oplus t$  takes  $p + q + O(k)$  time. Similarly, the computations for occurrences of (TR-CONST), (TR-APPG), and (TR-APP) take  $O(k)$ ,  $O(\ell + k)$ , and  $O(\ell(\ell + k))$  time, respectively. Thus, the running time for every node of the derivation tree is  $O(A^2)$ .

Hence, for each rule  $F \tilde{y} \tilde{x} \rightarrow t$  of  $\mathcal{G}$ , the computation of  $y_1 : \kappa_1, \dots, y_m : \kappa_m; x_1, \dots, x_k : \mathfrak{o} \vdash_{\mathcal{N}} t : \mathfrak{o} \rightsquigarrow (t_0, t_1, \dots, t_k, t_{k+1})$  takes  $O(A^2|t|)$  time. Also, that of all  $\tilde{y}_i$  and  $\tilde{y}'_i$  ( $1 \leq i \leq m$ ) takes  $O(A^2)$  time. Therefore, that of (TR-RULE) runs in  $O(A^2|t|)$  time. Thus the computation of  $\mathcal{R}'$  takes  $O(A^2|\mathcal{G}|_{\text{cy}})$  time.

Also the computation of  $\mathcal{N}^\ddagger$  takes  $O(|\mathcal{N}|)$  time because, for  $\kappa = \kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o}$ , that of  $\kappa^\ddagger$  (and simultaneously of  $\kappa^{\ddagger'}$ ) takes  $O(|\kappa_1| + \dots + |\kappa_k| + k + \ell)$  time. Hence, the computation of (TR-GRAM) runs in  $O(A^2|\mathcal{G}|_{\text{cy}} + |\mathcal{N}|)$ .  $\blacktriangleleft$

## B Proofs for Section 5.3

Due to the space limit, we focus only on some important cases when we perform case analysis in proofs. See the full version for detailed proofs.

First we show the substitution lemma (Lemma 19), where we use the following lemma:

► **Lemma 22 (Weakening).** *If  $\mathcal{K}; x_1, \dots, x_k : \mathfrak{o} \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_{\ell+k+1})$  where  $\text{gar}(\kappa) = \ell$ , then  $\mathcal{K}; x_0, x_1, \dots, x_k : \mathfrak{o} \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_\ell, t_{\ell+k+1}, t_{\ell+1}, \dots, t_{\ell+k}, t_{\ell+k+1})$ .*

**Proof.** Straightforward induction on  $t$ .  $\blacktriangleleft$

**Proof of Lemma 19.** First we show the second item of the lemma. We have:

$$\theta_0 u_{\text{gar}(\kappa') + |\bar{z}| + 1} = \theta_{k+1} u_{\text{gar}(\kappa') + |\bar{z}| + 1} = \theta_{k+1} u_0$$

where the former equation follows from Lemma 10, and the latter one can be shown by straightforward induction on  $u$  and case analysis on the last rule used for the derivation  $\tilde{y} : \tilde{\kappa}; \tilde{z} : \mathfrak{o} \vdash u : \kappa' \rightsquigarrow (u_0, \dots, u_{\text{gar}(\kappa') + |\bar{z}| + 1})$ , where we unfold the definition of  $\theta_{k+1}$  only in the case of (TR-VAR).

Next we show the first item, again by induction on  $u$  and case analysis on the last rule used for the derivation  $\tilde{y} : \tilde{\kappa}; \tilde{z} : \mathfrak{o} \vdash u : \kappa' \rightsquigarrow (u_0, \dots, u_{\text{gar}(\kappa') + |\bar{z}| + 1})$ .

■ Case of (TR-VAR): Let the last rule be the following:

$$\frac{\mathcal{K}(y_{i'}) = \tilde{\kappa} \Rightarrow \mathfrak{o}^{\ell_{i'}} \rightarrow \mathfrak{o}}{\tilde{y} : \tilde{\kappa}; \tilde{z} : \mathfrak{o} \vdash y_{i'} : \tilde{\kappa} \Rightarrow \mathfrak{o}^{\ell_{i'}} \rightarrow \mathfrak{o} \rightsquigarrow (y_{i',0}, y_{i',1}, \dots, y_{i',\ell_{i'}}, (y_{i',\ell_{i'}+1})^{|\bar{z}|+1})}$$

Now  $[\tilde{s}'/\tilde{y}]u = [\tilde{s}'/\tilde{y}]y_{i'} = s'_{i'}$ , and by the assumption and the weakening lemma

(Lemma 22),

$$\begin{aligned} \tilde{z}, \tilde{x} : \circ \vdash s'_{i'} : \kappa_{i'} &\rightsquigarrow \\ (s'_{i',0}, \dots, s'_{i',\ell_{i'}}, (s'_{i',\ell_{i'}+k+1})^{|\tilde{z}|}, s'_{i',\ell_{i'}+1}, \dots, s'_{i',\ell_{i'}+k}, s'_{i',\ell_{i'}+k+1}) \\ &= (\theta_0 y_{i',0}, \dots, \theta_0 y_{i',\ell_{i'}}, (\theta_0 y_{i',\ell_{i'}+1})^{|\tilde{z}|}, \theta_1 y_{i',0}, \dots, \theta_k y_{i',0}, \theta_0 y_{i',\ell_{i'}+1}) \end{aligned}$$

as required.

- Case of (TR-APP): Let the last rule be the following:

$$\frac{\begin{array}{l} \tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash s : \kappa'_0 \rightarrow \tilde{\kappa}' \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{\ell+|\tilde{z}|+1}) \\ \tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash t : \kappa'_0 \rightsquigarrow (t_0, \dots, t_{\ell+|\tilde{z}|+1}) \quad \mathbf{gar}(\kappa'_0) = \ell' \end{array}}{\tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash st : \kappa' \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow} \\ (s_0(t_0, \dots, t_{\ell+|\tilde{z}|+1}), s_1(t_1, \dots, t_{\ell+|\tilde{z}|+1}), \dots, s_{\ell}(t_1, \dots, t_{\ell+|\tilde{z}|+1}), \\ s_{\ell+1}(t_{\ell+1}, t_1, \dots, t_{\ell+|\tilde{z}|+1}), \dots, s_{\ell+|\tilde{z}|+1}(t_{\ell+|\tilde{z}|+1}, t_1, \dots, t_{\ell+|\tilde{z}|+1}))$$

By induction hypothesis and (TR-APP), we have:

$$\begin{array}{l} \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]s : \kappa'_0 \rightarrow \tilde{\kappa}' \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (\theta_0 s_0, \dots, \theta_0 s_{\ell+|\tilde{z}|}, \theta_1 s_0, \dots, \theta_k s_0, \theta_0 s_{\ell+|\tilde{z}|+1}) \\ \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]t : \kappa'_0 \rightsquigarrow (\theta_0 t_0, \dots, \theta_0 t_{\ell+|\tilde{z}|}, \theta_1 t_0, \dots, \theta_k t_0, \theta_0 t_{\ell+|\tilde{z}|+1}) \quad \mathbf{gar}(\kappa'_0) = \ell' \end{array} \\ \hline \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]s [\tilde{s}'/\tilde{y}]t : \kappa' \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow \\ (\theta_0 s_0(\theta_0 t_0, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \\ \theta_0 s_1(\theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \dots, \theta_0 s_{\ell}(\theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \\ \theta_0 s_{\ell+1}(\theta_0 t_{\ell+1}, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \dots, \theta_0 s_{\ell+|\tilde{z}|}(\theta_0 t_{\ell+|\tilde{z}|}, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \\ \theta_1 s_0(\theta_1 t_0, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \dots, \theta_k s_0(\theta_k t_0, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}), \\ \theta_0 s_{\ell+|\tilde{z}|+1}(\theta_0 t_{\ell+|\tilde{z}|+1}, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}))$$

Then it is enough to show

$$\begin{aligned} \theta_j s_0(\theta_j t_0, \theta_0 t_1, \dots, \theta_0 t_{\ell+|\tilde{z}|+1}) &= \theta_j (s_0(t_0, t_1, \dots, t_{\ell+|\tilde{z}|+1})) \quad (j \in \{1, \dots, k\}) \\ \text{i.e., } \theta_0 t_i &= \theta_j t_i \quad (i \in \{1, \dots, \ell', \ell' + |\tilde{z}| + 1\}, j \in \{1, \dots, k\}) \end{aligned}$$

which follows from Lemma 10.

- Case of (TR-APPG): Let the last rule be the following:

$$\frac{\begin{array}{l} \tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash s : \circ^{\ell+1} \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{\ell+|\tilde{z}|+2}) \\ \tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash t : \circ \rightsquigarrow (t_0, \dots, t_{|\tilde{z}|+1}) \end{array}}{\tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash st : \circ^\ell \rightarrow \circ \rightsquigarrow} \\ (s_0 \oplus (\mathbf{br} s_1 t_0), s_2, \dots, s_{\ell+1}, s_{\ell+2} \oplus (\mathbf{br} s_1 t_1), \dots, s_{\ell+|\tilde{z}|+2} \oplus (\mathbf{br} s_1 t_{|\tilde{z}|+1}))$$

By induction hypothesis and (TR-APPG), we have:

$$\begin{array}{l} \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]s : \circ^{\ell+1} \rightarrow \circ \rightsquigarrow (\theta_0 s_0, \dots, \theta_0 s_{\ell+|\tilde{z}|+1}, \theta_1 s_0, \dots, \theta_k s_0, \theta_0 s_{\ell+|\tilde{z}|+2}) \\ \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]t : \circ \rightsquigarrow (\theta_0 t_0, \dots, \theta_0 t_{|\tilde{z}|}, \theta_1 t_0, \dots, \theta_k t_0, \theta_0 t_{|\tilde{z}|+1}) \end{array} \\ \hline \tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}'/\tilde{y}]s [\tilde{s}'/\tilde{y}]t : \circ^\ell \rightarrow \circ \rightsquigarrow \\ (\theta_0 s_0 \oplus (\mathbf{br} \theta_0 s_1 \theta_0 t_0), \theta_0 s_2, \dots, \theta_0 s_{\ell+1}, \\ \theta_0 s_{\ell+2} \oplus (\mathbf{br} \theta_0 s_1 \theta_0 t_1), \dots, \theta_0 s_{\ell+|\tilde{z}|+1} \oplus (\mathbf{br} \theta_0 s_1 \theta_0 t_{|\tilde{z}|}), \\ \theta_1 s_0 \oplus (\mathbf{br} \theta_0 s_1 \theta_1 t_0), \dots, \theta_k s_0 \oplus (\mathbf{br} \theta_0 s_1 \theta_k t_0), \\ \theta_0 s_{\ell+|\tilde{z}|+2} \oplus (\mathbf{br} \theta_0 s_1 \theta_0 t_{|\tilde{z}|+1}))$$

Then it is enough to show  $\theta_0 s_1 = \theta_j s_1$  for  $j = 1, \dots, k$ , which follows from Lemma 10. ◀

Now we prove the subject reduction lemma:

**Proof of Lemma 20.** We restate Lemma 20 as follows with different meta-variables  $s', t', s'_i, t'_i$  for convenience of the proof:

If  $x_1, \dots, x_k : \circ \vdash s' : \circ \rightsquigarrow (s'_0, \dots, s'_{k+1})$  and  $s' \rightarrow_{\text{es}} t'$ , then there exist  $t'_0, \dots, t'_{k+1}$  such that  $x_1, \dots, x_k : \circ \vdash t' : \circ \rightsquigarrow (t'_0, \dots, t'_{k+1})$  and  $s'_i \approx t'_i$  for each  $i \in \{1, \dots, k+1\}$ .

The proof proceeds by induction on the derivation of  $x_1, \dots, x_k : \circ \vdash s' : \circ \rightsquigarrow (s'_0, \dots, s'_{k+1})$ .

Let  $s'$  be of the form  $E[s'']$  where  $s''$  is the redex of  $\rightarrow_{\text{es}}$ . The case where  $E \neq []$  can be proved easily by induction hypothesis (as the transformation rules are compositional). So we consider only the case where  $E = []$ . We perform case analysis on the redex  $s'' (= s')$ :

■ Case where  $s' = F \tilde{s} \tilde{t} \rightarrow_{\text{es}} \{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u$ : In this case,

$$F \tilde{y} \tilde{z} \rightarrow u \in \mathcal{R} \quad \mathcal{N}(F) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \quad |\tilde{\kappa}| = |\tilde{y}| = |\tilde{s}| \quad \ell = |\tilde{z}| = |\tilde{t}|,$$

and  $\tilde{z}$  do not occur in  $F \tilde{s} \tilde{t}$ .

By the derivation of  $\tilde{x} : \circ \vdash (s' =) F \tilde{s} \tilde{t} : \circ \rightsquigarrow (s'_0, \dots, s'_{k+1})$ , we have:

$$\begin{aligned} \tilde{x} : \circ \vdash F : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (F_0, F_1, \dots, F_\ell, (F_0)^{k+1}) \\ \tilde{x} : \circ \vdash s_i : \kappa_i \rightsquigarrow (s_{i,0}, \dots, s_{i,\ell_i+k+1}) \quad \text{gar}(\kappa_i) = \ell_i \quad (i = 1, \dots, |\tilde{s}|) \end{aligned} \quad (4)$$

$$\tilde{x} : \circ \vdash t_i : \circ \rightsquigarrow (t_{i,0}, \dots, t_{i,k+1}) \quad (i = 1, \dots, |\tilde{t}|) \quad (5)$$

$$(s'_0, \dots, s'_{k+1}) = (v_{\ell+1,0}, \dots, v_{\ell+1,k+1})$$

where for  $i = 1, \dots, |\tilde{s}|$  and  $j = 1, \dots, |\tilde{t}| (= \ell)$ , we define:

$$\begin{aligned} (v'_{1,0}, \dots, v'_{1,\ell+k+1}) &:= (F_0, F_1, \dots, F_\ell, (F_0)^{k+1}) \\ (v'_{i+1,0}, \dots, v'_{i+1,\ell+k+1}) &:= \\ &\quad (v'_{i,0}(s_{i,0}, \dots, s_{i,\ell_i}, s_{i,\ell_i+k+1}), \\ &\quad v'_{i,1}(s_{i,1}, \dots, s_{i,\ell_i}, s_{i,\ell_i+k+1}), \dots, v'_{i,\ell}(s_{i,1}, \dots, s_{i,\ell_i}, s_{i,\ell_i+k+1}), \\ &\quad v'_{i,\ell+1}(s_{i,\ell_i+1}, s_{i,1}, \dots, s_{i,\ell_i}, s_{i,\ell_i+k+1}), \dots, v'_{i,\ell+k+1}(s_{i,\ell_i+k+1}, s_{i,1}, \dots, s_{i,\ell_i}, s_{i,\ell_i+k+1})) \\ (v_{1,0}, \dots, v_{1,\ell+k+1}) &:= (v'_{|\tilde{s}|+1,0}, \dots, v'_{|\tilde{s}|+1,\ell+k+1}) \\ (v_{j+1,0}, \dots, v_{j+1,\ell+k+1-j}) &:= \\ &\quad (v_{j,0} \oplus (\text{br } v_{j,1} t_{j,0}), v_{j,2}, \dots, v_{j,\ell+1-j}, \\ &\quad v_{j,\ell+2-j} \oplus (\text{br } v_{j,1} t_{j,1}), \dots, v_{j,\ell+k+2-j} \oplus (\text{br } v_{j,1} t_{j,k+1})). \end{aligned}$$

Then we have

$$\begin{aligned} (s'_0, s'_1, \dots, s'_{k+1}) &= (v_{\ell+1,0}, v_{\ell+1,1}, \dots, v_{\ell+1,k+1}) \\ &= \left( v_{\ell,0} \oplus (\text{br } v_{\ell,1} t_{\ell,0}), v_{\ell,2} \oplus (\text{br } v_{\ell,1} t_{\ell,1}), \dots, v_{\ell,k+2} \oplus (\text{br } v_{\ell,1} t_{\ell,k+1}) \right) \\ &= \left( (v_{\ell-1,0} \oplus (\text{br } v_{\ell-1,1} t_{\ell-1,0})) \oplus (\text{br } v_{\ell-1,2} t_{\ell,0}), \right. \\ &\quad (v_{\ell-1,3} \oplus (\text{br } v_{\ell-1,1} t_{\ell-1,1})) \oplus (\text{br } v_{\ell-1,2} t_{\ell,1}), \dots, \\ &\quad \left. (v_{\ell-1,k+3} \oplus (\text{br } v_{\ell-1,1} t_{\ell-1,k+1})) \oplus (\text{br } v_{\ell-1,2} t_{\ell,k+1}) \right) \quad (6) \\ &= \dots \\ &= \left( v_{1,0} \oplus (\text{br } v_{1,1} t_{1,0}) \oplus \dots \oplus (\text{br } v_{1,\ell} t_{\ell,0}), \right. \\ &\quad v_{1,\ell+1} \oplus (\text{br } v_{1,1} t_{1,1}) \oplus \dots \oplus (\text{br } v_{1,\ell} t_{\ell,1}), \dots, \\ &\quad \left. v_{1,\ell+k+1} \oplus (\text{br } v_{1,1} t_{1,k+1}) \oplus \dots \oplus (\text{br } v_{1,\ell} t_{\ell,k+1}) \right) \end{aligned}$$

and

$$\begin{aligned}
v_{1,0} &= F_0(s_{1,0}, \dots, s_{1,\ell_1}, s_{1,\ell_1+k+1}) \cdots (s_{|\bar{s}|,0}, \dots, s_{|\bar{s}|,\ell_{|\bar{s}|}}, s_{|\bar{s}|,\ell_{|\bar{s}|}+k+1}) \\
v_{1,j} &= F_j(s_{1,1}, \dots, s_{1,\ell_1}, s_{1,\ell_1+k+1}) \cdots (s_{|\bar{s}|,1}, \dots, s_{|\bar{s}|,\ell_{|\bar{s}|}}, s_{|\bar{s}|,\ell_{|\bar{s}|}+k+1}) \quad (j = 1, \dots, \ell) \\
v_{1,\ell+j} &= F_0(s_{1,\ell_1+j}, s_{1,1}, \dots, s_{1,\ell_1}, s_{1,\ell_1+k+1}) \cdots (s_{|\bar{s}|,\ell_{|\bar{s}|}+j}, s_{|\bar{s}|,1}, \dots, s_{|\bar{s}|,\ell_{|\bar{s}|}}, s_{|\bar{s}|,\ell_{|\bar{s}|}+k+1}) \\
&\quad (j = 1, \dots, k+1).
\end{aligned}$$

Next let us consider  $t' = \{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u$ . For some  $\tilde{u}$ , we have

$$\tilde{y} : \tilde{\kappa}; \tilde{z} : \circ \vdash u : \circ \rightsquigarrow (u_0, \dots, u_{\ell+1}). \quad (7)$$

By (7), (4), and the substitution lemma (Lemma 19), we have

$$\tilde{z}, \tilde{x} : \circ \vdash [\tilde{s}/\tilde{y}]u : \circ \rightsquigarrow (\theta_0 u_0, \theta_0 u_1, \dots, \theta_0 u_\ell, \theta_1 u_0, \dots, \theta_k u_0, \theta_0 u_{\ell+1}) \quad (8)$$

$$\theta_0 u_{\ell+1} = \theta_{k+1} u_0 \quad (9)$$

where

$$\begin{aligned}
\theta_j &= \theta_{1,j} \cdots \theta_{|\tilde{\kappa}|,j} \quad (j = 0, \dots, k) \\
\theta_{i,0} &= [s_{i,0}/y_{i,0}, \dots, s_{i,\ell_i}/y_{i,\ell_i}, s_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \quad (i = 1, \dots, |\tilde{\kappa}|) \\
\theta_{i,j} &= [s_{i,\ell_i+j}/y_{i,0}, \dots, s_{i,\ell_i}/y_{i,\ell_i}, s_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \quad (i = 1, \dots, |\tilde{\kappa}|, j = 1, \dots, k+1).
\end{aligned}$$

Further, by (5), (8), and (TR-SUB), we have

$$\tilde{x} : \circ \vdash (t' =) \{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u : \circ \rightsquigarrow (t'_0, \dots, t'_{k+1})$$

where

$$\begin{aligned}
(t'_0, t'_1, \dots, t'_k, t'_{k+1}) &:= \\
&((\theta_0 u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br}(\theta_0 u_i) t_{i,0}), \\
&(\theta_1 u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br}(\theta_0 u_i) t_{i,1}), \dots, (\theta_k u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br}(\theta_0 u_i) t_{i,k}), \\
&(\theta_0 u_{\ell+1}) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br}(\theta_0 u_i) t_{i,k+1})).
\end{aligned} \quad (10)$$

Now, by (TR-RULE), the transformed grammar has the following rules

$$F_0 \tilde{y}_1 \cdots \widetilde{y_{|\bar{y}|}} \rightarrow u_0, \quad F_j \tilde{y}_1' \cdots \widetilde{y_{|\bar{y}|}'} \rightarrow u_j \quad (j = 1, \dots, \ell)$$

where

$$\tilde{y}_i = (y_{i,0}, \dots, y_{i,\mathbf{gar}(\kappa_i)+1}) \quad \tilde{y}_i' = (y_{i,1}, \dots, y_{i,\mathbf{gar}(\kappa_i)+1}) \quad (i = 1, \dots, |\bar{y}|).$$

Then,

$$v_{1,0} \longrightarrow^* \theta_0 u_0 \quad (11)$$

$$v_{1,j} \longrightarrow^* \theta_0 u_j \quad (j = 1, \dots, \ell) \quad (12)$$

$$v_{1,\ell+j} \longrightarrow^* \theta_j u_0 \quad (j = 1, \dots, k) \quad (13)$$

$$v_{1,\ell+k+1} \longrightarrow^* \theta_{k+1} u_0 = \theta_0 u_{\ell+1} \quad (14)$$

## 22:22 Size-Preserving Translations from Word Grammars to Tree Grammars

where: the equation in (14) is just (9); and for (12) note that  $y_{1,0}, \dots, y_{|\tilde{\kappa}|,0}$  do not occur in  $u_j$  by Lemma 10. Then, we have

$$\begin{aligned}
 & (s'_0, s'_1, \dots, s'_k, s'_{k+1}) \\
 = & (v_{1,0} \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} v_{1,i} t_{i,0}), & (\because (6)) \\
 & v_{1,\ell+1} \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} v_{1,i} t_{i,1}), \dots, v_{1,\ell+k} \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} v_{1,i} t_{i,k}) \\
 & v_{1,\ell+k+1} \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} v_{1,i} t_{i,k+1})) \\
 \approx & ((\theta_0 u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} (\theta_0 u_i) t_{i,0}), & (\because (11)-(14)) \\
 & (\theta_1 u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} (\theta_0 u_i) t_{i,1}), \dots, (\theta_k u_0) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} (\theta_0 u_i) t_{i,k}), \\
 & (\theta_0 u_{\ell+1}) \oplus \bigoplus_{i=1}^{\ell} (\mathbf{br} (\theta_0 u_i) t_{i,k+1})) \\
 = & (t'_0, t'_1, \dots, t'_k, t'_{k+1}) & (\because (10))
 \end{aligned}$$

as required. ◀