On Word and Frontier Languages of Unsafe Higher-Order Grammars*†

Kazuyuki Asada[1] and Naoki Kobayashi[2]

1 **The University of Tokyo, Tokyo, Japan**
asada@kb.is.s.u-tokyo.ac.jp
2 **The University of Tokyo, Tokyo, Japan**
koba@is.s.u-tokyo.ac.jp

Abstract

Higher-order grammars are an extension of regular and context-free grammars, where non-terminals may take parameters. They have been extensively studied in 1980's, and restudied recently in the context of model checking and program verification. We show that the class of unsafe order-(n+1) word languages coincides with the class of frontier languages of unsafe order-n tree languages. We use intersection types for transforming an order-(n+1) word grammar to a corresponding order-n tree grammar. The result has been proved for safe languages by Damm in 1982, but it has been open for unsafe languages, to our knowledge. Various known results on higher-order grammars can be obtained as almost immediate corollaries of our result.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases intersection types, higher-order grammars

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.111

1 Introduction

Higher-order grammars are an extension of regular and context-free grammars, where non-terminals may take trees or (higher-order) functions on trees as parameters. They were extensively studied in the 1980's [6, 7, 8], and recently reinvestigated in the context of model checking [10, 17] and applied to program verification [11].

The present paper shows that the class of unsafe order-$(n+1)$ word languages coincides with the class of "frontier languages" of unsafe order-n tree languages. Here, the frontier of a tree is the sequence of symbols that occur in the leaves of the tree from left to right, and the frontier language of a tree language consists of the frontiers of elements of the tree language. The special case where $n = 0$ corresponds to the well-known fact that the frontier language of a regular tree language is a context-free language. The result has been proved by Damm [6] for grammars with the safety restriction (see [16] for a nice historical account of the safety restriction), but it has been open for unsafe grammars, to our knowledge.[1]

Damm's proof relied on the safety restriction (in particular, the fact that variable renaming is not required for safe grammars [3]) and does not apply (at least directly) to the case of unsafe grammars. We instead use intersection types to transform an order-$(n+1)$ word grammar $\mathcal{G}$ to an order-n tree grammar $\mathcal{G}'$ such that the frontier language of $\mathcal{G}'$ coincides

* A full version [2] of the paper is available at http://arxiv.org/abs/1604.01595.

† This work was supported by JSPS Kakenhi 23220001 and 15H05706.

[1] Kobayashi et al. [13] mentioned the result, referring to the paper under preparation: "On Unsafe Tree and Leaf Languages," which is actually the present paper.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 111; pp. 111:1–111:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with the language generated by $\mathcal{G}$. Intersection types have been used for recent other studies of higher-order grammars and model checking [11, 13, 12, 15, 19, 18, 14, 20]; our proof in the present paper provides even more evidence that intersection types are a versatile tool for studies of higher-order grammars. Compared with the previous work on intersection types for higher-order grammars, the technical novelties include: (i) our intersection types (used in Section 3) are mixtures of non-linear and linear intersection types and (ii) our type-based transformation involves global restructuring of terms. These points have made the correctness of the transformations non-trivial and delicate.

As stressed by Damm [6] at the beginning of his paper, the result will be useful for analyzing properties of higher-order languages by induction on the order of grammars. Our result allows properties on (unsafe) order-n languages to be reduced to those on order-$(n-1)$ tree languages, and then the latter may be studied by investigating those on the path languages of order-$(n-1)$ tree languages, which are order-$(n-1)$ word languages.

The rest of this paper is structured as follows. Section 2 reviews the definition of higher-order grammars, and states the main result. Sections 3 and 4 prove the result by providing the (two-step) transformations from order-$(n+1)$ word grammars to order-n tree grammars. Section 5 discusses applications of the result. Section 6 discusses related work and Section 7 concludes the paper. For the space restriction, we omit some details and proofs, which are found in the full version [2].

2 Preliminaries

This section defines higher-order grammars and the languages generated by them, and then explains the main result. Most of the following definitions follow those in [13].

A higher-order grammar consists of non-deterministic rewriting rules of the form $A \to t$, where A is a non-terminal and t is a simply-typed λ-term that may contain non-terminals and terminals (tree constructors).

▶ **Definition 1** (types and terms)**.** The set of *simple types*,[2] ranged over by κ, is given by: $\kappa ::= \mathtt{o} \mid \kappa_1 \to \kappa_2$. The order and arity of a simple type κ, written $\mathtt{order}(\kappa)$ and $\mathtt{ar}(\kappa)$, are defined respectively by:

$$\mathtt{order}(\mathtt{o}) = 0 \qquad \mathtt{order}(\kappa_1 \to \kappa_2) = \max(\mathtt{order}(\kappa_1) + 1, \mathtt{order}(\kappa_2))$$
$$\mathtt{ar}(\mathtt{o}) = 0 \qquad \mathtt{ar}(\kappa_1 \to \kappa_2) = 1 + \mathtt{ar}(\kappa_2)$$

The type $\mathtt{o}$ describes trees, and $\kappa_1 \to \kappa_2$ describes functions from κ_1 to κ_2. The set of *λ-terms*, ranged over by t, is defined by: $t ::= x \mid A \mid a \mid t_1\, t_2 \mid \lambda x : \kappa.t$. Here, x ranges over variables, A over symbols called non-terminals, and a over symbols called terminals. We assume that each terminal a has a fixed arity; we write Σ for the map from terminals to their arities. A term t is called an *applicative term* (or simply a *term*) if it does not contain λ-abstractions. A (simple) type environment $\mathcal{K}$ is a sequence of type bindings of the form $x : \kappa$ such that if $\mathcal{K}$ contains $x : \kappa$ and $x' : \kappa'$ in different positions then $x \neq x'$. In type environments, non-terminals are also treated as variables. A λ-term t has type κ under $\mathcal{K}$ if $\mathcal{K} \vdash_{\mathtt{ST}} t : \kappa$ is derivable from the following typing rules.

$$\frac{}{\mathcal{K}, x : \kappa, \mathcal{K}' \vdash_{\mathtt{ST}} x : \kappa} \qquad \frac{}{\mathcal{K} \vdash_{\mathtt{ST}} a : \underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{\Sigma(a)} \to \mathtt{o}}$$

[2] We sometimes call simple types *sorts* in this paper, to avoid confusion with intersection types introduced later for grammar transformations.

$$\frac{\mathcal{K} \vdash_{\mathtt{ST}} t_1 : \kappa_2 \to \kappa \qquad \mathcal{K} \vdash_{\mathtt{ST}} t_2 : \kappa_2}{\mathcal{K} \vdash_{\mathtt{ST}} t_1\, t_2 : \kappa} \qquad \frac{\mathcal{K},\, x : \kappa_1 \vdash_{\mathtt{ST}} t : \kappa_2}{\mathcal{K} \vdash_{\mathtt{ST}} \lambda x : \kappa_1.t : \kappa_1 \to \kappa_2}$$

We call t a (finite, Σ-ranked) *tree* if t is an applicative term consisting of only terminals, and $\vdash_{\mathtt{ST}} t : \mathtt{o}$ holds. We write $\mathbf{Tree}_\Sigma$ for the set of Σ-ranked trees, and use the meta-variable π for a tree.

We often omit type annotations and just write $\lambda x.t$ for $\lambda x : \kappa.t$. We consider below only well-typed λ-terms of the form $\lambda x_1.\cdots\lambda x_k.t$, where t is an applicative term. We are now ready to define higher-order grammars.

▶ **Definition 2** (higher-order grammar)**.** A *higher-order grammar* is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where (i) Σ is a ranked alphabet; (ii) $\mathcal{N}$ is a map from a finite set of non-terminals to their types; (iii) $\mathcal{R}$ is a finite set of *rewriting rules* of the form $A \to \lambda x_1.\cdots\lambda x_\ell.t$, where $\mathcal{N}(A) = \kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o}$, t is an applicative term, and $\mathcal{N}, x_1 : \kappa_1, \ldots, x_\ell : \kappa_\ell \vdash_{\mathtt{ST}} t : \mathtt{o}$ holds for some $\kappa_1, \ldots, \kappa_\ell$. (iv) S is a non-terminal called *the start symbol*, and $\mathcal{N}(S) = \mathtt{o}$. The *order* of a grammar $\mathcal{G}$, written $\mathtt{order}(\mathcal{G})$, is the largest order of the types of non-terminals. We sometimes write $\Sigma_\mathcal{G}, \mathcal{N}_\mathcal{G}, \mathcal{R}_\mathcal{G}, S_\mathcal{G}$ for the four components of $\mathcal{G}$.

For a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, the rewriting relation $\longrightarrow_\mathcal{G}$ is defined by:

$$\frac{(A \to \lambda x_1.\cdots\lambda x_k.t) \in \mathcal{R}}{A\, t_1 \cdots t_k \longrightarrow_\mathcal{G} [t_1/x_1, \ldots, t_k/x_k]t} \qquad \frac{t_i \longrightarrow_\mathcal{G} t_i' \qquad i \in \{1, \ldots, k\} \qquad \Sigma(a) = k}{a\, t_1 \cdots t_k \longrightarrow_\mathcal{G} a\, t_1 \cdots t_{i-1}\, t_i'\, t_{i+1} \cdots t_k}$$

Here, $[t_1/x_1, \ldots, t_k/x_k]t$ is the term obtained by substituting t_i for the free occurrences of x_i in t. We write $\longrightarrow_\mathcal{G}^*$ for the reflexive transitive closure of $\longrightarrow_\mathcal{G}$.

The *tree language generated by* $\mathcal{G}$, written $\mathcal{L}(\mathcal{G})$, is the set $\{\pi \in \mathbf{Tree}_{\Sigma_\mathcal{G}} \mid S \longrightarrow_\mathcal{G}^* \pi\}$. We call a grammar $\mathcal{G}$ a *word grammar* if all the terminal symbols have arity 1 except the special terminal $\mathtt{e}$, whose arity is 0. The *word language* generated by a word grammar $\mathcal{G}$, written $\mathcal{L}_\mathtt{w}(\mathcal{G})$, is $\{a_1 \cdots a_n \mid a_1(\cdots(a_n\, \mathtt{e})\cdots) \in \mathcal{L}(\mathcal{G})\}$. The frontier word of a tree π, written $\mathbf{leaves}(\pi)$, is the sequence of symbols in the leaves of π. It is defined inductively by: $\mathbf{leaves}(a) = a$ when $\Sigma(a) = 0$, and $\mathbf{leaves}(a\, \pi_1 \cdots \pi_k) = \mathbf{leaves}(\pi_1) \cdots \mathbf{leaves}(\pi_k)$ when $\Sigma(a) = k > 0$. The *frontier language* generated by $\mathcal{G}$, written $\mathcal{L}_\mathtt{leaf}(\mathcal{G})$, is the set: $\{\mathbf{leaves}(\pi) \mid S \longrightarrow_\mathcal{G}^* \pi \in \mathbf{Tree}_{\Sigma_\mathcal{G}}\}$. In our main theorem, we assume that there is a special nullary symbol $\mathtt{e}$ and consider $\mathtt{e} \in \mathcal{L}_\mathtt{leaf}(\mathcal{G})$ as the empty word ε; i.e., we consider $\mathcal{L}^\varepsilon_\mathtt{leaf}(\mathcal{G})$ defined by:

$$\mathcal{L}^\varepsilon_\mathtt{leaf}(\mathcal{G}) := (\mathcal{L}_\mathtt{leaf}(\mathcal{G}) \setminus \{\mathtt{e}\}) \cup \{\varepsilon \mid \mathtt{e} \in \mathcal{L}_\mathtt{leaf}(\mathcal{G})\}.$$

We note that the classes of order-0 and order-1 word languages coincide with those of regular and context-free languages respectively. We often write $A\, x_1 \cdots x_k \to t$ for the rule $A \to \lambda x_1.\cdots\lambda x_k.t$. When considering the frontier language of a tree grammar, we assume, without loss of generality, that the ranked alphabet Σ has a unique binary symbol $\mathtt{br}$, and that all the other terminals have arity 0.

▶ **Example 3.** Consider the order-2 (word) grammar $\mathcal{G}_1 = (\{\mathtt{a} : 1, \mathtt{b} : 1, \mathtt{e} : 0\}, \{S : \mathtt{o}, F : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o}, A : (\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o}), B : (\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o})\}, \mathcal{R}_1, S)$, where $\mathcal{R}_1$ consists of:

$$S \to F\, \mathtt{a} \qquad S \to F\, \mathtt{b} \qquad A\, f\, x \to \mathtt{a}(f\, x) \qquad B\, f\, x \to \mathtt{b}(f\, x),$$
$$F\, f \to f(f\, \mathtt{e}) \qquad F\, f \to F\, (A\, f) \qquad F\, f \to F\, (B\, f).$$

S is reduced, for example, as follows.

$$S \longrightarrow F\, \mathtt{b} \longrightarrow F\, (A\, \mathtt{b}) \longrightarrow (A\, \mathtt{b})(A\, \mathtt{b}\, \mathtt{e}) \longrightarrow \mathtt{a}\, (\mathtt{b}\, (A\, \mathtt{b}\, \mathtt{e})) \longrightarrow \mathtt{a}(\mathtt{b}\, (\mathtt{a}(\mathtt{b}\, \mathtt{e}))).$$

The word language $\mathcal{L}_{\mathtt{w}}(\mathcal{G}_1)$ is $\{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^+\}$.

Consider the order-1 (tree) grammar $\mathcal{G}_2 = (\{\mathtt{br}\!:\!2, \mathtt{a}\!:\!0, \mathtt{b}\!:\!0, \mathtt{e}\!:\!0\}, \{S\!:\!\mathtt{o}, F\!:\!\mathtt{o}\to\mathtt{o}\}, \mathcal{R}_2, S)$, where $\mathcal{R}_2$ consists of:

$$S \to F\,\mathtt{a} \qquad S \to F\,\mathtt{b} \qquad F\,f \to \mathtt{br}\,f\,f \qquad F\,f \to F(\mathtt{br}\,\mathtt{a}\,f) \quad F\,f \to F(\mathtt{br}\,\mathtt{b}\,f).$$

The frontier language $\mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}_2)$ coincides with $\mathcal{L}_{\mathtt{w}}(\mathcal{G}_1)$ above.

The following is the main theorem we shall prove in this paper.

▶ **Theorem 4.** *For any order-$(n+1)$ word grammar $\mathcal{G}$ ($n \geq 0$), there exists an order-n tree grammar $\mathcal{G}'$ such that $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$.*

The converse of the above theorem also holds:

▶ **Theorem 5.** *For any order-n tree grammar $\mathcal{G}'$ such that no word in $\mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$ contains $\mathtt{e}$, there exists a word grammar $\mathcal{G}$ of order at most $n+1$ such that $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$.*

Since the construction of $\mathcal{G}$ is easy, we sketch it here; For $n \geq 1$, the grammar $\mathcal{G}$ is obtained by (i) changing the arity of each nullary terminal a ($\neq \mathtt{e}$) to one, i.e., $\Sigma_{\mathcal{G}}(a) := 1$, (ii) replacing the terminal $\mathtt{e}$ with a new non-terminal E of type $\mathtt{o}\to\mathtt{o}$, defined by $E\,x \to x$, and also the unique binary terminal $\mathtt{br}$ with a new non-terminal Br of type $(\mathtt{o}\to\mathtt{o})\to(\mathtt{o}\to\mathtt{o})\to(\mathtt{o}\to\mathtt{o})$, defined by $Br\,f\,g\,x \to f(g\,x)$, (iii) applying η-expansion to the right hand side of each (original) rule to add an order-0 argument, and (iv) adding new start symbol S' with rule $S' \to S\,\mathtt{e}$. For example, given the grammar $\mathcal{G}_2$ above, the following grammar is obtained:

$$\begin{array}{l} S' \to S\,\mathtt{e} \qquad S\,x \to F\,\mathtt{a}\,x \qquad S\,x \to F\,\mathtt{b}\,x \\ F\,f\,x \to Br\,f\,f\,x \qquad F\,f\,x \to F(Br\,\mathtt{a}\,f)\,x \qquad F\,f\,x \to F(Br\,\mathtt{b}\,f)\,x \\ E\,x \to x \qquad Br\,f\,g\,x \to f(g\,x). \end{array}$$

Theorem 4 is proved by two-step grammar transformations, both of which are based on intersection types. In the first step, we transform an order-$(n+1)$ word grammar $\mathcal{G}$ to an order-n tree grammar $\mathcal{G}''$ such that $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}_{\mathtt{leaf}}(\mathcal{G}'')\!\uparrow_{\mathtt{e}}$, where $\mathcal{L}\!\uparrow_{\mathtt{e}}$ is the word language obtained from $\mathcal{L}$ by removing all the occurrences of the special terminal $\mathtt{e}$; that is, the frontier language of $\mathcal{G}''$ is almost the same as $\mathcal{L}_{\mathtt{w}}(\mathcal{G})$, except that the former may contain multiple occurrences of the special, dummy symbol $\mathtt{e}$. In the second step, we clean up the grammar to eliminate $\mathtt{e}$ (except that a singleton tree $\mathtt{e}$ may be generated when $\epsilon \in \mathcal{L}_{\mathtt{w}}(\mathcal{G})$). The first and second steps shall be formalized in Sections 3 and 4 respectively.

For the target of the transformations, we use the following extended terms, in which a *set* of terms may occur in an argument position:

$$\begin{array}{rl} u \text{ (extended terms)} & ::= x \mid A \mid a \mid u_0 U \mid \lambda x.u \\ U & ::= \{u_1, \ldots, u_k\}\ (k \geq 1). \end{array}$$

Here, $u_0\,u_1$ is interpreted as just a shorthand for $u_0\{u_1\}$. Intuitively, $\{u_1,\ldots,u_k\}$ is considered a non-deterministic choice $u_1 + \cdots + u_k$, which (lazily) reduces to u_i non-deterministically. The typing rules are extended accordingly by:

$$\frac{\mathcal{K} \vdash_{\mathtt{ST}} u_0 : \kappa_1 \to \kappa \qquad \mathcal{K} \vdash_{\mathtt{ST}} U : \kappa_1}{\mathcal{K} \vdash_{\mathtt{ST}} u_0\,U : \kappa} \qquad\qquad \frac{\mathcal{K} \vdash_{\mathtt{ST}} u_i : \kappa \text{ for each } i \in \{1,\ldots,k\}}{\mathcal{K} \vdash_{\mathtt{ST}} \{u_1,\ldots,u_k\} : \kappa}$$

An *extended higher-order grammar* is the same as a higher-order grammar, except that each rewriting rule in $\mathcal{R}$ may be of the form $\lambda x_1 \cdots \lambda x_\ell.u$, where u may be an applicative extended term. The reduction rule for non-terminals is replaced by:

$$\frac{(A \to \lambda x_1 \cdots \lambda x_k.u) \in \mathcal{R} \qquad u' \in [U_1/x_1, \ldots, U_k/x_k]u}{A\,U_1 \cdots U_k \longrightarrow_{\mathcal{G}} u'}$$

where the substitution θu is defined by:

$$\theta a = \{a\} \qquad \theta x = \begin{cases} \theta(x) & (\text{if } x \in dom(\theta)) \\ \{x\} & (\text{otherwise}) \end{cases}$$
$$\theta(u_0 U) = \{v(\theta U) \mid v \in \theta u_0\} \qquad \theta\{u_1, \ldots, u_k\} = \theta u_1 \cup \cdots \cup \theta u_k\,.$$

Also, the other reduction rule is replaced by the following two rules:

$$\frac{u \longrightarrow_{\mathcal{G}} u' \qquad i \in \{1, \ldots, k\} \qquad \Sigma(a) = k}{a\,U_1 \cdots U_{i-1}\,\{u\}\,U_{i+1} \cdots U_k \longrightarrow_{\mathcal{G}} a\,U_1 \cdots U_{i-1}\,\{u'\}\,U_{i+1} \cdots U_k}$$

$$\frac{u \in U_i \qquad U_i \text{ is not a singleton} \qquad i \in \{1, \ldots, k\} \qquad \Sigma(a) = k}{a\,U_1 \cdots U_k \longrightarrow_{\mathcal{G}} a\,U_1 \cdots U_{i-1}\,\{u\}\,U_{i+1} \cdots U_k}$$

Note that unlike in the extended grammar introduced in [13], there is no requirement that each u_i in $\{u_1, \ldots, u_k\}$ is used at least once. Thus, the extended syntax does not change the expressive power of grammars. A term set $\{u_1, \ldots, u_k\}$ can be replaced by $A\,x_1 \cdots x_\ell$ with the rewriting rules $A\,x_1 \cdots x_\ell \to u_i$, where $\{x_1, \ldots, x_\ell\}$ is the set of variables occurring in some of $u_1, \ldots, u_k$. In other words, for any order-n extended grammar $\mathcal{G}$, there is an (ordinary) order-n grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

3 Step 1: from order-$(n+1)$ grammars to order-n tree grammars

In this section, we show that for any order-$(n+1)$ grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ such that $\Sigma(\mathtt{e}) = 0$ and $\Sigma(a) = 1$ for every $a \in dom(\Sigma) \setminus \{\mathtt{e}\}$, there exists an order-$n$ grammar $\mathcal{G}'$ such that $\Sigma_{\mathcal{G}'} = \{\mathtt{br} \mapsto 2, \mathtt{e} \mapsto 0\} \cup \{a \mapsto 0 \mid \Sigma(a) = 1\}$ and $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}_{\mathtt{leaf}}(\mathcal{G}')\uparrow_{\mathtt{e}}$.

For technical convenience, we assume below that, for every type κ occurring in $\mathcal{N}_{\mathcal{G}}(A)$ for some A, if κ is of the form $\mathtt{o} \to \kappa'$, then $\mathtt{order}(\kappa') \leq 1$. This does not lose generality, since any function $\lambda x : \mathtt{o}.t$ of type $\mathtt{o} \to \kappa'$ with $\mathtt{order}(\kappa') > 1$ can be replaced by the term $\lambda x' : \mathtt{o} \to \mathtt{o}.[x'\mathtt{e}/x]t$ of type $(\mathtt{o} \to \mathtt{o}) \to \kappa'$ (without changing the order of the term), and any term t of type $\mathtt{o}$ can be replaced by the term $K\,t$ of type $\mathtt{o} \to \mathtt{o}$, where K is a non-terminal of type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$, with rule $K\,x\,y \to x$. See [2] for the details of this transformation.

The basic idea of the transformation is to remove all the order-0 arguments (i.e., arguments of tree type $\mathtt{o}$). This reduces the order of each term by 1; for example, terms of types $\mathtt{o} \to \mathtt{o}$ and $(\mathtt{o} \to \mathtt{o}) \to \mathtt{o}$ will respectively be transformed to those of types $\mathtt{o}$ and $\mathtt{o} \to \mathtt{o}$. Order-0 arguments can indeed be removed as follows. Suppose we have a term $t_1\,t_2$ where $t_1 : \mathtt{o} \to \mathtt{o}$. If t_1 does not use the order-0 argument t_2, then we can simply replace $t_1\,t_2$ with $t_1^{\#}$ (where $t_1^{\#}$ is the result of recursively applying the transformation to t_1). If t_1 uses the argument t_2, the word generated by $t_1\,t_2$ must be of the form $w_1 w_2$, where w_2 is generated by t_2; in other words, t_1 can only append a word to the word generated by t_2. Thus, $t_1\,t_2$ can be transformed to $\mathtt{br}\;t_1^{\#}\;t_2^{\#}$, which can generate a tree whose frontier coincides with $w_1 w_2$ (if $\mathtt{e}$ is ignored). As a special case, a constant word $\mathtt{a}\,\mathtt{e}$ can be transformed to $\mathtt{br}\;\mathtt{a}\;\mathtt{e}$. As a little more complex example, consider the term $A\,(\mathtt{b}\,\mathtt{e})$, where A is defined by $A\,x \to \mathtt{a}\,x$. Since A uses the argument, the term $A\,(\mathtt{b}\,\mathtt{e})$ is transformed to $\mathtt{br}\;A\;(\mathtt{br}\;\mathtt{b}\;\mathtt{e})$. Since A no longer takes an argument, we substitute $\mathtt{e}$ for x in the body of the rule for A (and apply the transformation recursively to $\mathtt{a}\,\mathtt{e}$). The resulting rule for A is: $A \to \mathtt{br}\;\mathtt{a}\;\mathtt{e}$. Thus, the

term after the transformation generates the tree $\mathtt{br}\,(\mathtt{br}\,\mathtt{a}\,\mathtt{e})\,(\mathtt{br}\,\mathtt{b}\,\mathtt{e})$. Its frontier word is $\mathtt{aebe}$, which is equivalent to the word $\mathtt{ab}$ generated by the original term, up to removals of $\mathtt{e}$; recall that redundant occurrences of $\mathtt{e}$ will be removed by the second transformation. Note that the transformation sketched above depends on whether each order-0 argument is actually used or not. Thus, we introduce intersection types to express such information, and define the transformation as a type-directed one.

Simple types are refined to the following intersection types.

$$\delta ::= \mathtt{o} \mid \sigma \to \delta \qquad \sigma ::= \delta_1 \wedge \cdots \wedge \delta_k \quad (k \geq 0)$$

We write $\top$ for $\delta_1 \wedge \cdots \wedge \delta_k$ when $k = 0$. We assume some total order $<$ on intersection types, and require that $\delta_1 < \cdots < \delta_k$ whenever $\delta_1 \wedge \cdots \wedge \delta_k$ occurs in an intersection type. Intuitively, $(\delta_1 \wedge \cdots \wedge \delta_k) \to \delta$ describes a function that uses an argument according to types $\delta_1, \ldots, \delta_k$, and the returns a value of type δ. As a special case, the type $\top \to \mathtt{o}$ describes a function that ignores an argument, and returns a tree. Thus, according to the idea of the transformation sketched above, if x has type $\top \to \mathtt{o}$, $x\,t$ would be transformed to x; if x has type $\mathtt{o} \to \mathtt{o}$, $x\,t$ would be transformed to $\mathtt{br}\;x\;t^{\#}$. In the last example above, the type $\mathtt{o} \to \mathtt{o}$ should be interpreted as a function that uses the argument *just once*; otherwise the transformation to $\mathtt{br}\;x\;t^{\#}$ would be incorrect. Thus, the type $\mathtt{o}$ should be treated as a linear type, for which weakening and dereliction are disallowed. In contrast, we need not enforce, for example, that a value of the intersection type $\mathtt{o} \to \mathtt{o}$ should be used just once. Therefore, we classify intersection types into two kinds; one called *balanced*, which may be treated as non-linear types, and the other called *unbalanced*, which must be treated as linear types. For that purpose, we introduce two refinement relations $\delta ::_{\mathrm{b}} \kappa$ and $\delta ::_{\mathrm{u}} \kappa$; the former means that δ is a balanced intersection type of sort κ, and the latter means that δ is an unbalanced intersection type of sort κ. The relations are defined as follows, by mutual induction; k may be 0.

$$\frac{\delta_j ::_{\mathrm{u}} \kappa \qquad j \in \{1, \ldots, k\} \qquad \delta_i ::_{\mathrm{b}} \kappa \ (\text{for each } i \in \{1, \ldots, k\} \setminus \{j\})}{\delta_1 \wedge \cdots \wedge \delta_k ::_{\mathrm{u}} \kappa} \qquad \frac{\delta_i ::_{\mathrm{b}} \kappa \ (\text{for each } i \in \{1, \ldots, k\})}{\delta_1 \wedge \cdots \wedge \delta_k ::_{\mathrm{b}} \kappa}$$

$$\frac{}{\mathtt{o} ::_{\mathrm{u}} \mathtt{o}} \qquad \frac{\sigma ::_{\mathrm{b}} \kappa \qquad \delta ::_{\mathrm{u}} \kappa'}{\sigma \to \delta ::_{\mathrm{u}} \kappa \to \kappa'} \qquad \frac{\sigma ::_{\mathrm{u}} \kappa \qquad \delta ::_{\mathrm{u}} \kappa'}{\sigma \to \delta ::_{\mathrm{b}} \kappa \to \kappa'} \qquad \frac{\sigma ::_{\mathrm{b}} \kappa \qquad \delta ::_{\mathrm{b}} \kappa'}{\sigma \to \delta ::_{\mathrm{b}} \kappa \to \kappa'}$$

A type δ is called *balanced* if $\delta ::_{\mathrm{b}} \kappa$ for some κ, and called *unbalanced* if $\delta ::_{\mathrm{u}} \kappa$ for some κ. Intuitively, unbalanced types describe trees or closures that contain the end of a word (i.e., symbol $\mathtt{e}$). Intersection types that are neither balanced nor unbalanced are considered ill-formed, and excluded out. For example, the type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$ (as an intersection type) is ill-formed; since $\mathtt{o}$ is unbalanced, $\mathtt{o} \to \mathtt{o}$ must also be unbalanced according to the rules for arrow types, but it is actually balanced. Note that, in fact, no term can have the intersection type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$ in a word grammar. We write $\delta :: \kappa$ if $\delta ::_{\mathrm{b}} \kappa$ or $\delta ::_{\mathrm{u}} \kappa$.

We introduce a type-directed transformation relation $\Gamma \vdash t : \delta \Rightarrow u$ for terms, where Γ is a set of type bindings of the form $x : \delta$, called a *type environment*, t is a source term, and u is the image of the transformation, which may be an extended term. We write $\Gamma_1 \cup \Gamma_2$ for the union of Γ_1 and Γ_2; it is defined only if, whenever $x : \delta \in \Gamma_1 \cap \Gamma_2$, δ is balanced. In other words, unbalanced types are treated as linear types, whereas balanced ones as non-linear (or idempotent) types. We write $\mathbf{bal}(\Gamma)$ if δ is balanced for every $x : \delta \in \Gamma$.

The relation $\Gamma \vdash t : \delta \Rightarrow u$ is defined inductively by the following rules.

$$\frac{\mathbf{bal}(\Gamma)}{\Gamma, x : \delta \vdash x : \delta \Rightarrow x_\delta} \ (\textsc{Tr1-Var}) \qquad \frac{A :: \mathcal{N}(A) \qquad \mathbf{bal}(\Gamma)}{\Gamma \vdash A : \delta \Rightarrow A_\delta} \ (\textsc{Tr1-NT})$$

$$\frac{\mathbf{bal}(\Gamma)}{\Gamma \vdash \mathtt{e} : \mathtt{o} \Rightarrow \mathtt{e}} \quad (\textsc{Tr1-Const0}) \qquad \frac{\Sigma(a) = 1 \qquad \mathbf{bal}(\Gamma)}{\Gamma \vdash a : \mathtt{o} \to \mathtt{o} \Rightarrow a} \quad (\textsc{Tr1-Const1})$$

$$\frac{\begin{array}{c}\Gamma_0 \vdash s : \delta_1 \wedge \cdots \wedge \delta_k \to \delta \Rightarrow v \\ \Gamma_i \vdash t : \delta_i \Rightarrow U_i \text{ and } \delta_i \neq \mathtt{o} \text{ (for each } i \in \{1, \ldots, k\})\end{array}}{\Gamma_0 \cup \Gamma_1 \cup \cdots \cup \Gamma_k \vdash st : \delta \Rightarrow v U_1 \cdots U_k} \quad (\textsc{Tr1-App1})$$

$$\frac{\Gamma_0 \vdash s : \mathtt{o} \to \delta \Rightarrow V \qquad \Gamma_1 \vdash t : \mathtt{o} \Rightarrow U}{\Gamma_0 \cup \Gamma_1 \vdash st : \delta \Rightarrow \mathtt{br}\, V\, U} \quad (\textsc{Tr1-App2})$$

$$\frac{\Gamma \vdash t : \delta \Rightarrow u_i \text{ (for each } i \in \{1, \ldots, k\}) \qquad k \geq 1}{\Gamma \vdash t : \delta \Rightarrow \{u_1, \ldots, u_k\}} \quad (\textsc{Tr1-Set})$$

$$\frac{\begin{array}{c}\Gamma, x : \delta_1, \ldots, x : \delta_k \vdash t : \delta \Rightarrow u \qquad x \notin dom(\Gamma) \\ \delta_i \neq \mathtt{o} \text{ for each } i \in \{1, \ldots, k\}\end{array}}{\Gamma \vdash \lambda x.t : \delta_1 \wedge \cdots \wedge \delta_k \to \delta \Rightarrow \lambda x_{\delta_1} \cdots \lambda x_{\delta_k}.u} \quad (\textsc{Tr1-Abs1})$$

$$\frac{\Gamma, x : \mathtt{o} \vdash t : \delta \Rightarrow u}{\Gamma \vdash \lambda x.t : \mathtt{o} \to \delta \Rightarrow [\mathtt{e}/x_{\mathtt{o}}]u} \quad (\textsc{Tr1-Abs2})$$

In rule (Tr1-Var), a variable is replicated for each type. This is because the image of the transformation of a term substituted for x is different depending on the type of the term; accordingly, in rule (Tr1-Abs1), bound variables are also replicated, and in rule (Tr1-App1), arguments are replicated. In rule (Tr1-NT), a non-terminal is also replicated for each type. In rules (Tr1-Const0) and (Tr1-Const1), constants are mapped to themselves; however, the arities of all the constants become 0. In these rules, Γ may contain only bindings on balanced types.

In rule (Tr1-App1), the first premise indicates that the function s uses the argument t according to types $\delta_1, \ldots, \delta_k$. Since the image of the transformation of t depends on its type, we replicate the argument to $U_1, \ldots, U_k$. For each type δ_i, the result of the transformation is not unique (but finite); thus, we represent the image of the transformation as a *set* U_i of terms. (Recall the remark at the end of Section 2 that a set of terms can be replaced by an ordinary term by introducing auxiliary non-terminals.) For example, consider a term $A(x\,y)$. It can be transformed to $A_{\delta_1 \to \delta}\{x_{\delta_0 \to \delta_1} y_{\delta_0}, x_{\delta_0' \to \delta_1} y_{\delta_0'}\}$ under the type environment $\{x : \delta_0 \to \delta_1, x : \delta_0' \to \delta_1, y : \delta_0, y : \delta_0'\}$. Note that k in rule (Tr1-App1) (and also (Tr1-Abs1)) may be 0, in which case the argument disappears in the image of the transformation.

In rule (Tr1-App2), as explained at the beginning of this section, the argument t of type $\mathtt{o}$ is removed from s and instead attached as a sibling node of the tree generated by (the transformation image of) s. Accordingly, in rule (Tr1-Abs2), the binder for x is removed and x in the body of the abstraction is replaced with the empty tree $\mathtt{e}$. In rule (Tr1-Set), type environments are shared. This is because $\{u_1, \ldots, u_k\}$ represents the choice $u_1 + \cdots + u_k$; unbalanced (i.e. linear) values should be used in the same manner in $u_1, \ldots, u_k$.

The transformation rules for rewriting rules and grammars are given by:

$$\frac{\emptyset \vdash \lambda x_1. \cdots \lambda x_k.t : \delta \Rightarrow \lambda x_1'. \cdots \lambda x_\ell'.u \qquad \delta :: \mathcal{N}(A)}{(A\, x_1\, \cdots\, x_k \to t) \Rightarrow (A_\delta\, x_1'\, \cdots\, x_\ell' \to u)} \quad (\textsc{Tr1-Rule})$$

$$\frac{\Sigma' = \{\mathtt{br} \mapsto 2, \mathtt{e} \mapsto 0\} \cup \{a \mapsto 0 \mid \Sigma(a) = 1\} \quad \mathcal{N}' = \{A_\delta : [\![\delta :: \kappa]\!] \mid \mathcal{N}(A) = \kappa \wedge \delta :: \kappa\} \quad \mathcal{R}' = \{r' \mid \exists r \in \mathcal{R}. r \Rightarrow r'\}}{(\Sigma, \mathcal{N}, \mathcal{R}, S) \Rightarrow (\Sigma', \mathcal{N}', \mathcal{R}', S_{\mathtt{o}})} \quad (\textsc{Tr1-Gram})$$

Here, $[\![\delta :: \kappa]\!]$ is defined by:

$$[\![\delta :: \kappa]\!] = \mathtt{o} \qquad (\text{if } \mathtt{order}(\kappa) \leq 1)$$
$$[\![(\delta_1 \wedge \cdots \wedge \delta_k \to \delta) :: (\kappa_0 \to \kappa)]\!] = [\![\delta_1 :: \kappa_0]\!] \to \ldots \to [\![\delta_k :: \kappa_0]\!] \to [\![\delta :: \kappa]\!] \qquad (\text{if } \mathtt{order}(\kappa_0 \to \kappa) > 1)$$

▶ **Example 6.** Recall the grammar $\mathcal{G}_1$ in Example 3. For the term $\lambda f.\lambda x.\mathtt{a}(f\,x)$ of the rule for A, we have the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \mathtt{a} : \mathtt{o} \to \mathtt{o} \Rightarrow \mathtt{a}}\ \textsc{Const1} \quad \dfrac{\dfrac{}{f : \mathtt{o} \to \mathtt{o} \vdash f : \mathtt{o} \to \mathtt{o} \Rightarrow f_{\mathtt{o}\to\mathtt{o}}}\ \textsc{Var} \quad \dfrac{}{x : \mathtt{o} \vdash x : \mathtt{o} \Rightarrow x_{\mathtt{o}}}\ \textsc{Var}}{f : \mathtt{o} \to \mathtt{o}, x : \mathtt{o} \vdash f\,x : \mathtt{o} \Rightarrow \mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, x_{\mathtt{o}}}\ \textsc{App2}}{f : \mathtt{o} \to \mathtt{o}, x : \mathtt{o} \vdash \mathtt{a}(f\,x) : \mathtt{o} \Rightarrow \mathtt{br}\,\mathtt{a}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, x_{\mathtt{o}})}\ \textsc{App2}}{f : \mathtt{o} \to \mathtt{o} \vdash \lambda x.\mathtt{a}(f\,x) : \mathtt{o} \to \mathtt{o} \Rightarrow \mathtt{br}\,\mathtt{a}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, \mathtt{e})}\ \textsc{Abs2}}{\emptyset \vdash \lambda f.\lambda x.\mathtt{a}(f\,x) : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \Rightarrow \lambda f_{\mathtt{o}\to\mathtt{o}}.\mathtt{br}\,\mathtt{a}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, \mathtt{e})}\ \textsc{Abs1}$$

Notice that the argument x has been removed, and the result of the transformation has type $\mathtt{o} \to \mathtt{o}$. The whole grammar is transformed to the grammar consisting of the following rules.

$$S_{\mathtt{o}} \to F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\,\mathtt{a} \qquad S_{\mathtt{o}} \to F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\,\mathtt{b}$$
$$A_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to \mathtt{br}\,\mathtt{a}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, \mathtt{e}) \qquad B_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to \mathtt{br}\,\mathtt{b}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, \mathtt{e})$$
$$F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to \mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\,(\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, \mathtt{e}) \qquad F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}(A_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}})$$
$$F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}(B_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}}).$$

Here, we have omitted rules that are unreachable from $S_{\mathtt{o}}$. For example, the rule

$$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}}\, f_{\top\to\mathtt{o}}\, f_{\mathtt{o}\to\mathtt{o}} \to \mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, f_{\top\to\mathtt{o}}$$

may be obtained from the following derivation, but it is unreachable from $S_{\mathtt{o}}$, since F is never called with an argument of type $(\top \to \mathtt{o}) \wedge (\mathtt{o} \to \mathtt{o})$.

$$\dfrac{\dfrac{\dfrac{}{f : \mathtt{o} \to \mathtt{o} \vdash f \Rightarrow f_{\mathtt{o}\to\mathtt{o}}}\ \textsc{Var} \quad \dfrac{\dfrac{}{f : \top \to \mathtt{o} \vdash f : \top \to \mathtt{o} \Rightarrow f_{\top\to\mathtt{o}}}\ \textsc{Var}}{f : \top \to \mathtt{o} \vdash f\,\mathtt{e} : \mathtt{o} \Rightarrow f_{\top\to\mathtt{o}}}\ \textsc{App1}}{f : \top \to \mathtt{o}, f : \mathtt{o} \to \mathtt{o} \vdash f(f\,\mathtt{e}) : \mathtt{o} \Rightarrow \mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, f_{\top\to\mathtt{o}}}\ \textsc{App2}}{\emptyset \vdash \lambda f.f(f\,\mathtt{e}) : (\top \to \mathtt{o}) \wedge (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \Rightarrow \lambda f_{\top\to\mathtt{o}}.\lambda f_{\mathtt{o}\to\mathtt{o}}.\mathtt{br}\, f_{\mathtt{o}\to\mathtt{o}}\, f_{\top\to\mathtt{o}}}\ \textsc{Abs1}$$

The following theorem states the correctness of the first transformation.

▶ **Theorem 7.** *Let $\mathcal{G}$ be an order-$(n+1)$ word grammar. If $\mathcal{G} \Rightarrow \mathcal{G}''$, then $\mathcal{G}''$ is an (extended) grammar of order at most n. Furthermore, $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}_{\mathtt{leaf}}(\mathcal{G}'')\uparrow_{\mathtt{e}}$.*

4 Step 2: removing dummy symbols

We now describe the second step for eliminating redundant symbols $\mathtt{e}$, which have been introduced by (Tr1-Abs2). By the remark at the end of Section 2, we assume that the result of the first transformation is an ordinary grammar, not containing extended terms. We also assume that $\mathtt{br}$ occurs only in the fully applied form. This does not lose generality, because otherwise we can replace $\mathtt{br}$ by a new non-terminal Br and add the rule $Br\,x\,y \to \mathtt{br}\,x\,y$.

The idea of the transformation is to use intersection types to distinguish between terms that generate trees consisting of only $\mathtt{br}$ and $\mathtt{e}$, and those that generate trees containing other arity-0 terminals. We assign the type $\mathtt{o}_\epsilon$ to the former terms, and $\mathtt{o}_+$ to the latter. A term $\mathtt{br}\,t_0\,t_1$ is transformed to (i) $\mathtt{br}\,t_0^\#\,t_1^\#$ if both t_0 and t_1 have type $\mathtt{o}_+$ (where $t_i^\#$ is the image of the transformation of t_i), (ii) $t_i^\#$ if t_i has type $\mathtt{o}_+$ and t_{1-i} has type $\mathtt{o}_\epsilon$, and (iii) $\mathtt{e}$ if both t_0 and t_1 have type $\mathtt{o}_\epsilon$. As in the transformation of the previous section, we replicate each non-terminal and variable for each intersection type. For example, the nonterminal $A : \mathtt{o} \to \mathtt{o}$ defined by $A\,x \to x$ would be replicated to $A_{\mathtt{o}_+\to\mathtt{o}_+}$ and $A_{\mathtt{o}_\epsilon\to\mathtt{o}_\epsilon}$.

We first define the set of intersection types by:

$$\xi ::= \mathtt{o}_\epsilon \mid \mathtt{o}_+ \mid \xi_1 \wedge \cdots \wedge \xi_k \to \xi$$

We assume some total order $<$ on intersection types, and require that whenever we write $\xi_1 \wedge \cdots \wedge \xi_k$, $\xi_1 < \cdots < \xi_k$ holds. We define the refinement relation $\xi :: \kappa$ inductively by: (i) $\mathtt{o}_\epsilon :: \mathtt{o}$, (ii) $\mathtt{o}_+ :: \mathtt{o}$, and (iii) $(\xi_1 \wedge \cdots \wedge \xi_k \to \xi) :: (\kappa_1 \to \kappa_2)$ if $\xi :: \kappa_2$ and $\xi_i :: \kappa_1$ for every $i \in \{1, \ldots, k\}$. We consider only types ξ such that $\xi :: \kappa$ for some κ. For example, we forbid an ill-formed type like $\mathtt{o}_+ \wedge (\mathtt{o}_+ \to \mathtt{o}_+) \to \mathtt{o}_+$.

We introduce a type-based transformation relation $\Xi \vdash t : \xi \Rightarrow u$, where Ξ is a type environment (i.e., a set of bindings of the form $x : \xi$), t is a source term, ξ is the type of t, and u is the result of transformation. The relation is defined inductively by the rules below.

$$\frac{}{\Xi, x : \xi \vdash x : \xi \Rightarrow x_\xi}\ (\textsc{Tr2-Var}) \qquad \frac{}{\Xi \vdash \mathtt{e} : \mathtt{o}_\epsilon \Rightarrow \mathtt{e}}\ (\textsc{Tr2-Const0}) \qquad \frac{\Sigma(a) = 0 \qquad a \neq \mathtt{e}}{\Xi \vdash a : \mathtt{o}_+ \Rightarrow a}\ (\textsc{Tr2-Const1})$$

$$\frac{\Xi \vdash t_0 : \xi_0 \Rightarrow u_0 \qquad \Xi \vdash t_1 : \xi_1 \Rightarrow u_1 \qquad (u, \xi) = \begin{cases} (\mathtt{br}\,u_0\,u_1, \mathtt{o}_+) & \text{if } \xi_0 = \xi_1 = \mathtt{o}_+ \\ (u_i, \mathtt{o}_+) & \text{if } \xi_i = \mathtt{o}_+ \text{ and } \xi_{1-i} = \mathtt{o}_\epsilon \\ (\mathtt{e}, \mathtt{o}_\epsilon) & \text{if } \xi_0 = \xi_1 = \mathtt{o}_\epsilon \end{cases}}{\Xi \vdash \mathtt{br}\,t_0\,t_1 : \xi \Rightarrow u}\ (\textsc{Tr2-Const2})$$

$$\frac{\xi :: \mathcal{N}(F) \qquad A\,x_1\,\cdots\,x_k \to t \in \mathcal{R} \qquad \emptyset \vdash \lambda x_1.\cdots\lambda x_k.t : \xi \Rightarrow \lambda y_1.\cdots\lambda y_\ell.u}{\Xi \vdash A : \xi \Rightarrow A_\xi}\ (\textsc{Tr2-NT})$$

$$\frac{\Xi \vdash s : \xi_1 \wedge \cdots \wedge \xi_k \to \xi \Rightarrow v \qquad \Xi \vdash t : \xi_i \Rightarrow U_i \text{ (for each } i \in \{1, \ldots, k\})}{\Xi \vdash st : \xi \Rightarrow vU_1 \cdots U_k}\ (\textsc{Tr2-App})$$

$$\frac{\Xi \vdash t : \xi \Rightarrow u_i \text{ (for each } i \in \{1, \ldots, k\}) \qquad k \geq 1}{\Xi \vdash t : \xi \Rightarrow \{u_1, \ldots, u_k\}}\ (\textsc{Tr2-Set})$$

$$\frac{\Xi, x : \xi_1, \ldots, x : \xi_k \vdash t : \xi \Rightarrow u}{\Xi \vdash \lambda x.t : \xi_1 \wedge \cdots \wedge \xi_k \to \xi \Rightarrow \lambda x_{\xi_1} \cdots \lambda x_{\xi_k}.u}\ (\textsc{Tr2-Abs})$$

The transformation of rewriting rules and grammars is defined by:

$$\frac{\emptyset \vdash \lambda x_1.\cdots\lambda x_k.t : \xi \Rightarrow \lambda x'_1.\cdots\lambda x'_\ell.t' \qquad \xi :: \mathcal{N}(A)}{(A \to \lambda x_1.\cdots\lambda x_k.t) \Rightarrow (A_\xi \to \lambda x'_1.\cdots\lambda x'_\ell.t')}\ (\textsc{Tr2-Rule})$$

$$\frac{\begin{array}{c}\mathcal{N}' = \{A_\xi : [\![\xi]\!] \mid \mathcal{N}(A) = \kappa \wedge \xi :: \kappa\} \\ \mathcal{R}' = \{r' \mid \exists r \in \mathcal{R}.r \Rightarrow r'\} \cup \{S' \to S_{\mathtt{o}_\epsilon}, S' \to S_{\mathtt{o}_+}\}\end{array}}{(\Sigma, \mathcal{N}, \mathcal{R}, S) \Rightarrow (\Sigma, \mathcal{N}', \mathcal{R}', S')}\ (\textsc{Tr2-Gram})$$

Here, $[\![\xi]\!]$ is defined by:

$$[\![\mathtt{o}_\epsilon]\!] = [\![\mathtt{o}_+]\!] = \mathtt{o} \qquad [\![\,\xi_1 \wedge \cdots \wedge \xi_k \to \xi]\!] = [\![\xi_1]\!] \to \cdots \to [\![\xi_k]\!] \to [\![\xi]\!]$$

We explain some key rules. In (Tr2-Var) we replicate a variable for each type, as in the first transformation. The rules (Tr2-Const0) and (Tr2-Const1) are for nullary constants, which are mapped to themselves. We assign type $\mathtt{o}_\epsilon$ to $\mathtt{e}$ and $\mathtt{o}_+$ to the other constants. The rule (Tr2-Const2) is for the binary tree constructor $\mathtt{br}$. As explained above, we eliminate terms that generate empty trees (those consisting of only $\mathtt{br}$ and $\mathtt{e}$). For example, if $\xi_0 = \mathtt{o}_\epsilon$ and $\xi_1 = \mathtt{o}_+$, then t_0 may generate an empty tree; thus, the whole term is transformed to u_1.

The rule (Tr2-NT) replicates a terminal for each type, as in the case of variables. The middle and rightmost premises require that there is some body t of A that can indeed be transformed according to type ξ. Without this condition, for example, A defined by the rule $A \to A$ would be transformed to $A_{\mathtt{o}_\epsilon}$ by $\emptyset \vdash A : \mathtt{o}_\epsilon \Rightarrow A_{\mathtt{o}_\epsilon}$, but $A_{\mathtt{o}_\epsilon}$ diverges and does not produce an empty tree. That would make the rule (Tr2-Const2) unsound: when a source term is $\mathtt{br}\,A\,\mathtt{a}$, it would be transformed to $\mathtt{a}$, but while the original term does not generate a tree, the result of the transformation does. In short, the two premises are required to ensure that whenever $\emptyset \vdash t : \mathtt{o}_\epsilon \Rightarrow u$ holds, t can indeed generate an empty tree. In (Tr2-App), the argument is replicated for each type. Unlike in the transformation in the previous section, type environments can be shared among the premises, since linearity does not matter here. The other rules for terms are analogous to those in the first transformation.

In rule (Tr2-Gram) for grammars, we prepare a start symbol S' and add the rules $S' \to S_{\mathtt{o}_\epsilon}, S' \to S_{\mathtt{o}_+}$. We remark that the rewriting rule for $S_{\mathtt{o}_\epsilon}$ (resp. $S_{\mathtt{o}_+}$) is generated only if the original grammar generates an empty (resp. non-empty) tree. For example, in the extreme case where $\mathcal{R} = \{S \to S\}$, we have $\mathcal{R}' = \{S' \to S_{\mathtt{o}_\epsilon}, S' \to S_{\mathtt{o}_+}\}$, without any rules to rewrite $S_{\mathtt{o}_\epsilon}$ or $S_{\mathtt{o}_+}$.

▶ **Example 8.** Let us consider the grammar $\mathcal{G}_3 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\mathcal{N} = \{S : \mathtt{o}, A : \mathtt{o} \to \mathtt{o}, B : \mathtt{o} \to \mathtt{o}, F : \mathtt{o} \to \mathtt{o}\}$, and $\mathcal{R}$ consists of:

$$S \to F\,\mathtt{a} \qquad S \to F\,\mathtt{b} \qquad A\,f \to \mathtt{br}\,\mathtt{a}\,(\mathtt{br}\,f\,\mathtt{e}) \qquad B\,f \to \mathtt{br}\,\mathtt{b}\,(\mathtt{br}\,f\,\mathtt{e})$$
$$F\,f \to \mathtt{br}\,f\,(\mathtt{br}\,f\,\mathtt{e}) \qquad F\,f \to F(A\,f) \qquad F\,f \to F(B\,f)$$

It is the same as the grammar obtained in Example 6, except that redundant subscripts on non-terminals and variables have been removed. The body of the rule for A is transformed as follows.

$$\dfrac{\dfrac{}{f : \mathtt{o}_+ \vdash \mathtt{a} : \mathtt{o}_+ \Rightarrow \mathtt{a}}\ \textsc{Const1} \qquad \dfrac{\dfrac{\dfrac{}{f : \mathtt{o}_+ \vdash f : \mathtt{o}_+ \Rightarrow f_{\mathtt{o}_+}}\ \textsc{Var} \quad \dfrac{}{f : \mathtt{o}_+ \vdash \mathtt{e} : \mathtt{o}_\epsilon \Rightarrow \mathtt{e}}\ \textsc{Const0}}{f : \mathtt{o}_+ \vdash \mathtt{br}\,f\,\mathtt{e} : \mathtt{o}_+ \Rightarrow f_{\mathtt{o}_+}}\ \textsc{Const2}}{f : \mathtt{o}_+ \vdash \mathtt{br}\,\mathtt{a}\,(\mathtt{br}\,f\,\mathtt{e}) : \mathtt{o}_+ \Rightarrow \mathtt{br}\,\mathtt{a}\,f_{\mathtt{o}_+}}\ \textsc{Const2}}{\emptyset \vdash \lambda f.\mathtt{br}\,\mathtt{a}\,(\mathtt{br}\,f\,\mathtt{e}) : \mathtt{o}_+ \to \mathtt{o}_+ \Rightarrow \lambda f_{\mathtt{o}_+}.\mathtt{br}\,\mathtt{a}\,f_{\mathtt{o}_+}}\ \textsc{Abs}$$

The whole rules are transformed to:

$$S' \to S_{\mathtt{o}_+} \qquad S' \to S_{\mathtt{o}_\epsilon} \qquad S_{\mathtt{o}_+} \to F_{\mathtt{o}_+ \to \mathtt{o}_+}\,\mathtt{a} \qquad S_{\mathtt{o}_+} \to F_{\mathtt{o}_+ \to \mathtt{o}_+}\,\mathtt{b}$$
$$A_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+} \to \mathtt{br}\,\mathtt{a}\,f_{\mathtt{o}_+} \qquad B_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+} \to \mathtt{br}\,\mathtt{b}\,f_{\mathtt{o}_+} \qquad F_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+} \to \mathtt{br}\,f_{\mathtt{o}_+}\,f_{\mathtt{o}_+}$$
$$F_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+} \to F_{\mathtt{o}_+ \to \mathtt{o}_+}(A_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+}) \qquad F_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+} \to F_{\mathtt{o}_+ \to \mathtt{o}_+}(B_{\mathtt{o}_+ \to \mathtt{o}_+}\,f_{\mathtt{o}_+})$$

Here, we have omitted rules on non-terminals unreachable from S'.

The following theorem claims the correctness of the transformation. The proof is given in [2]. The main theorem (Theorem 4) follows from Theorems 7, 9, and the fact that any order-m grammar with $m < n$ can be converted to an order-n grammar by adding a dummy non-terminal of order n.

▶ **Theorem 9.** *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be an order-n tree grammar. If $\mathcal{G} \Rightarrow \mathcal{G}'$, then $\mathcal{G}'$ is a tree grammar of order at most n, and $\mathcal{L}_{\mathtt{leaf}}(\mathcal{G})\uparrow_{\mathtt{e}} = \mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$.*

5 Applications

5.1 Unsafe order-2 word languages = safe order-2 word languages

As mentioned in Section 1, many of the earlier results on higher-order grammars [6, 10] were for the subclass called *safe* higher-order grammars. In safe grammars, the (simple) types of terms are restricted to *homogeneous types* [6] of the form $\kappa_1 \to \cdots \to \kappa_k \to \mathtt{o}$, where $\mathtt{order}(\kappa_1) \geq \cdots \geq \mathtt{order}(\kappa_k)$, and arguments of the same order must be supplied simultaneously. For example, if A has type $(\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o}) \to \mathtt{o}$, then the term $f\,(A\,f\,f)$ where $f : \mathtt{o} \to \mathtt{o}$ is valid, but $g\,(A\,f)$ where $g : ((\mathtt{o} \to \mathtt{o}) \to \mathtt{o}) \to \mathtt{o}, f : \mathtt{o} \to \mathtt{o}$ is not: the partial application $A\,f$ is disallowed, since A expects another order-1 argument. *Unsafe* grammars (which are just called higher-order grammars in the present paper) are higher-order grammars without the safety restriction.

For order-2 word languages, Aehlig et al. [1] have shown that safety is not a genuine restriction. Our result in the present paper provides an alternative, short proof. Given an unsafe order-2 word grammar $\mathcal{G}$, we can obtain an equivalent order-1 grammar $\mathcal{G}'$ such that $\mathcal{L}_{\mathtt{w}}(\mathcal{G}) = \mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$. Note that $\mathcal{G}'$ is necessarily safe, since it is order-1 and hence there are no partial applications. Now, apply the backward transformation sketched in Section 2 to obtain an order-2 word grammar $\mathcal{G}''$ such that $\mathcal{L}_{\mathtt{w}}(\mathcal{G}'') = \mathcal{L}^{\varepsilon}_{\mathtt{leaf}}(\mathcal{G}')$. By the construction of the backward transformation, $\mathcal{G}''$ is clearly a safe grammar: Since the type of each term occurring in $\mathcal{G}'$ is $\mathtt{o} \to \cdots \to \mathtt{o} \to \mathtt{o}$, the type of the corresponding term of $\mathcal{G}''$ is $(\mathtt{o} \to \mathtt{o}) \to \cdots \to (\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o})$. Since all the arguments of type $\mathtt{o}$ are applied simultaneously in $\mathcal{G}'$, all the arguments of type $\mathtt{o} \to \mathtt{o}$ are also applied simultaneously in $\mathcal{G}''$. Thus, for any unsafe order-2 word grammar, there exists an equivalent safe order-2 word grammar.

5.2 Diagonal problem

The diagonal problem [5] asks, given a (word or tree) language L and a set S of symbols, whether for all n, there exists $w_n \in L$ such that $\forall a \in S. |w_n|_a \geq n$. Here, $|w|_a$ denotes the number of occurrences of a in w. A decision algorithm for the diagonal problem can be used for computing downward closures [21], which in turn have applications to program verification. Hague et al. [9] recently showed that the diagonal problem is decidable for safe higher-order word languages, and Clemente et al. [4] extended the result for unsafe tree languages. For the single letter case of the diagonal problem (where $|S| = 1$), we can obtain an alternative proof as follows. First, following the approach of Hague et al. [9], we can use logical reflection to reduce the single letter diagonal problem for an unsafe order-n tree language to that for the path language of an unsafe order-n tree language. We can then use our transformation to reduce the latter to the single letter diagonal problem for an unsafe order-$(n-1)$ tree language.

5.3 Context-sensitivity of order-3 word languages

By using the result of this paper and the context-sensitivity of order-2 tree languages [13], we can prove that any order-3 word language is context-sensitive, i.e., the membership problem for an order-3 word language can be decided in non-deterministic linear space. Given an order-3 word grammar $\mathcal{G}$, we first construct a corresponding order-2 tree grammar $\mathcal{G}'$ in

advance. Given a word w, we can construct a tree π whose frontier word is w one by one, and check whether $\pi \in \mathcal{L}(\mathcal{G}')$. Since the size of π is linearly bounded by the length $|w|$ of w, $\pi \stackrel{?}{\in} \mathcal{L}(\mathcal{G}')$ can be checked in space linear with respect to $|w|$. Thus, $w \in \mathcal{L}_{\mathtt{w}}(\mathcal{G})$ can be decided in non-deterministic linear space (with respect to the size of w).

6 Related Work

As already mentioned in Section 1, higher-order grammars have been extensively studied in 1980's [6, 7, 8], but most of those results have been for safe grammars. In particular, Damm [6] has shown an analogous result for safe grammars, but his proof does not extend to the unsafe case.

As also mentioned in Section 1, intersection types have been used in recent studies of (unsafe) higher-order grammars. In particular, type-based transformations of grammars and λ-terms have been studied in [14, 13, 4]. Clement et al. [4], independently from ours, gave a transformation from an order-$(n+1)$ "narrow" tree language (which subsumes a word language as a special case) to an order-n tree language; this transformation preserves the number of occurrences of each symbol in each tree. When restricted to word languages, our result is stronger in that our transformation is guaranteed to preserve the order of symbols as well, and does not add any additional leaf symbols (though they are introduced in the intermediate step); consequently, our proofs are more involved. They use different intersection types, but the overall effect of their transformation seems similar to that of our first transformation. Thus, it may actually be the case that their transformation also preserves the order of symbols, although they have not proved so.

7 Conclusion

We have shown that for any unsafe order-$(n+1)$ word grammar $\mathcal{G}$, there exists an unsafe order-n tree grammar $\mathcal{G}'$ whose frontier language coincides with the word language $\mathcal{L}_{\mathtt{w}}(\mathcal{G})$. The proof is constructive in that we provided (two-step) transformations that indeed construct $\mathcal{G}'$ from $\mathcal{G}$. The transformations are based on a combination of linear/non-linear intersection types, which may be interesting in its own right. As Damm [6] suggested, we expect the result to be useful for further studies of higher-order languages; in fact, we have discussed a few applications of the result.

Acknowledgments. We would like to thank Takeshi Tsukada for helpful discussions and thank Pawel Parys for information about the related work [4].

References

1 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety is not a restriction at level 2 for string languages. In *Proceedings of FoSSaCS 2005*, volume 3441 of *LNCS*, pages 490–504. Springer, 2005.

2 Kazuyuki Asada and Naoki Kobayashi. On word and frontier languages of unsafe higher-order grammars. *CoRR*, abs/1604.01595, 2016.

3 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009.

4 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recusion schemes is decidable. In *Proceedings of LICS 2016*, 2016.

5 Wojciech Czerwinski and Wim Martens. A note on decidable separability by piecewise testable languages. *CoRR*, abs/1410.1042, 2014. URL: `http://arxiv.org/abs/1410.1042`.

6 Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.

7 Joost Engelfriet. Iterated stack automata and complexity classes. *Info. Comput.*, 95(1):21–75, 1991.

8 Joost Engelfriet and Heiko Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Inf.*, 26(1/2):131–192, 1988.

9 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proceedings of POPL 2016*, pages 151–163, 2016. `doi:10.1145/2837614.2837627`.

10 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *LNCS*, pages 253–267. Springer, 2001.

11 Naoki Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013.

12 Naoki Kobayashi. Pumping by typing. In *Proceedings of LICS 2013*, pages 398–407. IEEE Computer Society, 2013.

13 Naoki Kobayashi, Kazuhiro Inaba, and Takeshi Tsukada. Unsafe order-2 tree languages are context-sensitive. In *Proceedings of FoSSaCS 2014*, volume 8412 of *LNCS*, pages 149–163. Springer, 2014.

14 Naoki Kobayashi, Kazutaka Matsuda, Ayumi Shinohara, and Kazuya Yaguchi. Functional programs as compressed data. *Higher-Order and Symbolic Computation*, 2013.

15 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

16 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015.

17 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.

18 Pawel Parys. How many numbers can a lambda-term contain? In *Proceedings of FLOPS 2014*, volume 8475 of *LNCS*, pages 302–318. Springer, 2014. `doi:10.1007/978-3-319-07151-0_19`.

19 Sylvain Salvati and Igor Walukiewicz. Typing weak MSOL properties. In Andrew M. Pitts, editor, *Proceedings of FoSSaCS 2015*, volume 9034 of *LNCS*, pages 343–357. Springer, 2015. `doi:10.1007/978-3-662-46678-0_22`.

20 Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over böhm trees. In *Proceedings of CSL-LICS'14*, pages 78:1–78:10. ACM, 2014. `doi:10.1145/2603088.2603133`.

21 Georg Zetzsche. An approach to computing downward closures. In *Proceedings of ICALP 2015*, volume 9135 of *LNCS*, pages 440–451. Springer, 2015. `doi:10.1007/978-3-662-47666-6_35`.