Unsafe Order-2 Tree Languages are Context-Sensitive

Naoki Kobayashi[1], Kazuhiro Inaba[2], and Takeshi Tsukada[3]

[1] The University of Tokyo
[2] Google Inc.
[3] University of Oxford and JSPS Postdoctoral Fellow for Research Abroad

Abstract. Higher-order grammars have been extensively studied in 1980's and interests in them have revived recently in the context of higher-order model checking and program verification, where higher-order grammars are used as models of higher-order functional programs. A lot of theoretical questions remain open, however, for *unsafe* higher-order grammars (grammars without the so-called safety condition). In this paper, we show that any tree languages generated by order-2 unsafe grammars are context-sensitive. This also implies that any unsafe order-3 word languages are context-sensitive. The proof involves novel technique based on typed lambda-calculus, such as type-based grammar transformation.

1 Introduction

Higher-order (or high-level) grammars, where non-terminal symbols may take higher-order functions as arguments, have been introduced in 1970's [19, 20, 15] and extensively studied in 1980's [3]. They form a natural extension of Chomsky hierarchy [20], in the sense that they form an infinite language hierarchy, where the order-0 and order-1 word languages are exactly regular languages and context-free languages respectively. Recently, higher-order grammars have been studied as models of higher-order programs [8, 16], and applied to automated verification of higher-order programs [9, 13, 17].

Earlier theoretical results on higher-order grammars [3, 8, 6] have been for those with the so-called *safety* restriction [8] (or, with the condition on *derived types* [3]). Although some of the analogous results have recently been obtained for *unsafe* grammars (those without the safety restriction) [16, 7, 14], many problems still remain open, such as the context-sensitiveness of higher-order languages. This is a pity, as many of the recent applications of higher-order grammars make use of unsafe ones.

In the present paper, we are interested in the open problem mentioned above: whether the languages generated by higher-order grammars are context-sensitive. As a solution to a special case of the open problem, we show that the tree languages (or more precisely, the word languages obtained by preorder traversal of trees, because the context-sensitiveness is usually the terminology for word languages) generated by any order-2 grammars are also context-sensitive. Since the

order-$(n+1)$ word languages can be obtained as the leaf languages of trees generated by order-n grammars [11], the result also implies that the word languages generated by order-3 grammars are context-sensitive.[1]

Our techniques to prove the context-sensitiveness of order-2 tree languages are quite different from those used in Inaba and Maneth's proof for context-sensitiveness of safe languages [6]. Recall that the context-sensitiveness is equivalent to the membership problem being NLIN-SPACE (non-deterministic linear space). To show that, Inaba and Maneth decomposed higher-order (safe) transducers (whose image is the set of higher-order safe languages) into macro tree transducers, and transformed the transducers so that the size of intermediate trees increases monotonically. For the unsafe case, similar decomposition appears to be extremely difficult.

Instead of going through transducers or automata, we directly reason about grammars with a help of techniques of typed λ-calculus (intersection types, in particular). The high-level structure of our proof is actually similar to that of the (straightforward) proof of the context-sensitivity of context-free languages. For a context-free grammar (say, $\{S \to \mathtt{a}AA, A \to \epsilon \mid \mathtt{a}A\mathtt{b}\}$), one can eliminate ϵ-rules ($A \to \epsilon$ in the above example) to ensure that the size of intermediate phrases occurring in a production of a final word w is bounded by the size of w. For example, the above grammar can be transformed to $\{S \to \mathtt{a}AA \mid \mathtt{a} \mid \mathtt{a}A, A \to \mathtt{a}A\mathtt{b} \mid \mathtt{ab}\}$, by propagating information that A may be replaced by ϵ. The first part of our proof shows that intersection types can be used to achieve a similar (but more elaborate) transformation of higher-order grammars to exclude out certain rewriting rules. More precisely, given a finite set $\mathcal{C}$ of functions, one can exclude out rules that allow non-terminals to behave like one of the functions in $\mathcal{C}$. The second part of the proof shows that for the order-2 case, if we choose as $\mathcal{C}$ a set of "permutator [2]-like" terms, then the size of intermediate terms occurring in a production of a tree π is linearly bounded by the size of π. Thus, given an order-2 grammar $\mathcal{G}$, one can first transform $\mathcal{G}$ to an equivalent grammar $\mathcal{G}'$ that satisfies the property above, and then the membership of a tree π in the tree language of $\mathcal{G}'$ can be decided in space linear in π. This implies that the language of (word representation of) trees generated by $\mathcal{G}$ is context-sensitive.

From a practical viewpoint, the result may be applicable to the following problem: given a program P and a possible execution trace (or an execution tree) π, is π a real trace of P? If P is a simply-typed program with recursion and finite base types, one can use the technique of [9] to construct a grammar that represents all the possible traces of P. One can then use the above algorithm to decide the membership problem in linear space with respect to the size of π. If one asks many questions for a fixed P and different π, using the above algorithm is theoretically more efficient than using higher-order model checking [9].

The rest of the paper is structured as follows. Section 2 defines higher-order grammars and the languages generated by grammars. Section 3 describes the

[1] The order-2 word languages are known to be context-sensitive. The result follows from context-sensitiveness of *safe* word languages [6] and the equivalence of safe and unsafe word languages for the order-2 case [1].

type-based grammar transformation that removes certain rewriting rules. Section 4 focuses on order-2 grammars and shows that after the grammar transformation, the size of intermediate terms is linearly bounded by the size of the produced tree. Section 5 discusses related work and Section 6 concludes. For the space limitation, we omit some details and proofs, which are found in an extended version of this paper, available from the first author's web page.

2 Preliminaries

This section defines higher-order grammars and the languages generated by them. When f is a map, we write $\mathit{dom}(f)$ and $\mathit{codom}(f)$ for the domain and codomain of f.

Definition 1 (types). *The set of* ***simple types****, ranged over by κ, is defined 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 by:*

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

Intuitively, $\mathtt{o}$ is the type of trees. We assume a ranked alphabet Σ, which is a map from a finite set of symbols (called **terminals**) to their arities. We use each terminal a as a tree constructor of arity $\Sigma(a)$. We assume a finite set of symbols called **non-terminals**, ranged over by A.

Definition 2 (λ-terms). *The set of λ-****terms****, ranged over by t, is defined by: $t ::= x \mid A \mid a \mid t_1\, t_2 \mid \lambda x{:}\kappa.t$. A term t is called an* ***applicative term*** *(or simply a* ***term****) if it does not contain λ-abstractions.*

We often omit the type annotation and just write $\lambda x.t$ for $\lambda x : \kappa.t$. We consider only well-typed terms; the type judgment relation $\mathcal{K} \vdash_{\mathtt{ST}} t : \kappa$ (where non-terminals are treated as variables) is defined inductively by:

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

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

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

Definition 3 (higher-order grammar). *A* ***higher-order grammar*** *(called simply a* ***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 $A\, x_1 \cdots x_\ell \to t$, where $A \in \mathit{dom}(\mathcal{N})$*

*and t is an applicative term. We require that $\mathcal{N}(A)$ must be of the form $\kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o}$ and $\mathcal{N}, x_1 : \kappa_1, \ldots, x_\ell : \kappa_\ell \vdash_{\mathtt{ST}} t : \mathtt{o}$ must hold. (iv) S is a non-terminal called **the start symbol**, and $\mathcal{N}(S) = \mathtt{o}$. The **order** (**arity**, resp.) of a grammar $\mathcal{G}$, written $\mathtt{order}(\mathcal{G})$ ($\mathtt{ar}(\mathcal{G})$, resp.), is the largest order (arity, resp.) 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:

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

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 $\{\pi \in \mathbf{Tree}_{\Sigma_\mathcal{G}} \mid S \longrightarrow_\mathcal{G}^* \pi\}$. When the arity of every symbol in Σ is at most 1, the **word language** generated by $\mathcal{G}$ is $\{a_1 \cdots a_n \mid a_1(\cdots(a_n\, e)\cdots) \in \mathcal{L}(\mathcal{G})\}$. The **leaf language** generated by $\mathcal{G}$, written $\mathcal{L}_\mathtt{leaf}(\mathcal{G})$, is the set: $\{\boldsymbol{leaves}(\pi) \mid S \longrightarrow_\mathcal{G}^* \pi \in \mathbf{Tree}_{\Sigma_\mathcal{G}}\}$, where $\boldsymbol{leaves}(\pi)$ is the sequence of symbols in the leaves of π, defined inductively by: $\boldsymbol{leaves}(a) = a$, and $\boldsymbol{leaves}(a\, \pi_1\, \pi_2) = \boldsymbol{leaves}(\pi_1)\boldsymbol{leaves}(\pi_2)$. The **order of a tree language** is the smallest order of a grammar that generates the language.*

*A grammar is **safe** if for the type $\kappa_1 \to \cdots \to \cdots \to \kappa_\ell \to \mathtt{o}$ of each term t, (i) $\mathtt{order}(\kappa_1) \geq \cdots \geq \mathtt{order}(\kappa_\ell)$ holds, and (ii) if $\mathtt{order}(\kappa_i) = \mathtt{order}(\kappa_{i+1})$, the i-th and $(i+1)$-th arguments of t are passed always together. Grammars without the safety restriction are sometimes called **unsafe**, to emphasize the fact that there is no safety restriction. (Thus, the set of unsafe grammars include safe grammars.) A language is called **safe** if it is generated by some safe grammar.*

In the rest of this paper, we assume that every terminal has arity 0 or 2. This does not lose generality, because every tree can be represented by a corresponding binary tree with linear size increase.

Example 1. Consider the order-2 grammar $\mathcal{G}_0 = (\{\mathtt{a}:2, \mathtt{b}:2, \mathtt{e}:0\}, \{S:\mathtt{o}, F:(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}, C:(\mathtt{o}\to\mathtt{o}\to\mathtt{o})\to(\mathtt{o}\to\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}, T:(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of the rules:

$$\begin{array}{lll} S \to F\,(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} & F\,g\,x \to g\,x, & F\,g\,x \to F\,(T\,g)\,x \\ C\,g\,h\,x \to g\,x\,x & C\,g\,h\,x \to h\,x\,x & T\,g\,x \to g(g\,x). \end{array}$$

Then, the following is a possible reduction sequence:

$$\begin{array}{l} S \longrightarrow F\,(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} \longrightarrow F\,(T\,(C\,\mathtt{a}\,\mathtt{b}))\,\mathtt{e} \longrightarrow T\,(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} \\ \longrightarrow (C\,\mathtt{a}\,\mathtt{b})\,(C\,\mathtt{a}\,\mathtt{b}\,\mathtt{e}) \longrightarrow \mathtt{a}\,(C\,\mathtt{a}\,\mathtt{b}\,\mathtt{e})\,(C\,\mathtt{a}\,\mathtt{b}\,\mathtt{e}) \longrightarrow^* \mathtt{a}\,(\mathtt{b}\,\mathtt{e}\,\mathtt{e})\,(\mathtt{a}\,\mathtt{e}\,\mathtt{e}). \end{array}$$

$\mathcal{L}(\mathcal{G}_0)$ is the set of perfect finite trees of height 2^n (where all the leaves have the same depth). $\mathcal{L}_\mathtt{leaf}(\mathcal{G}_0) = \{\mathtt{e}^{2^{2^n}} \mid n \geq 0\}$.

Example 2. Consider the grammar $\mathcal{G}_1 = (\{\mathtt{f}{:}2, \mathtt{g}{:}2, \mathtt{a}{:}0, \mathtt{b}{:}0, \mathtt{e}{:}0\}, \{S{:}\mathtt{o}, F{:}(\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \to \mathtt{o}, G : \mathtt{o} \to \mathtt{o}, H : \mathtt{o} \to \mathtt{o}\}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of:

$$S \to F\,G\,\mathtt{a}\,\mathtt{b} \qquad F\,\varphi\,x\,y \to \mathtt{f}(F(F\,\varphi\,x)\,y\,(H\,y))(\mathtt{f}\,(\varphi\,y)\,x) \qquad F\,\varphi\,x\,y \to \mathtt{e}$$
$$G\,x \to \mathtt{g}\,x\,\mathtt{e} \qquad H\,x \to \mathtt{g}\,\mathtt{e}\,x.$$

This has been obtained from the grammar conjectured to be inherently unsafe ([8], p.213), by adding the rule $F\,\varphi\,x\,y \to \mathtt{e}$ (so that the grammar generates a set of finite trees, instead of an infinite tree) and encoding unary tree constructors $\mathtt{g}$ and $\mathtt{h}$ in their grammar as G and H (so that $\mathtt{h}(\pi)$ and $\mathtt{g}(\pi)$ are represented by $\mathtt{g}\,\mathtt{e}\,\pi'$ and $\mathtt{g}\,\pi'\,\mathtt{e}$ respectively). The following is a possible reduction sequence:

$$S \longrightarrow F\,G\,\mathtt{a}\,\mathtt{b} \longrightarrow \mathtt{f}(F(F\,G\,\mathtt{a})\,\mathtt{b}\,(H\,\mathtt{b}))(\mathtt{f}\,(G\,\mathtt{b})\,\mathtt{a}) \longrightarrow \mathtt{f}\,\mathtt{e}\,(\mathtt{f}\,(G\,\mathtt{b})\,\mathtt{a})$$
$$\longrightarrow \mathtt{f}\,\mathtt{e}\,(\mathtt{f}\,(\mathtt{g}\,\mathtt{b}\,\mathtt{e})\,\mathtt{a}).$$

3 Type-Based Grammar Transformation

As mentioned in Section 1, a key idea of our proof is to first transform a grammar to an equivalent grammar, so that the size of intermediate terms in a production sequence of tree π is linearly bound by the size of π. Note that the size of intermediate terms is not bounded for arbitrary grammars. For example, for the rewriting rules $\{S \to F\,\mathtt{e}, F\,x \to \mathtt{e}, F\,x \to F(F\,x)\}$, an arbitrarily large intermediate term $F^n\,\mathtt{e}$ may occur in a production of $\mathtt{e}$. As another example, replace the rule $F\,x \to \mathtt{e}$ above with $F\,x \to x$. Again, an arbitrarily large intermediate term $F^n\,\mathtt{e}$ may occur in a production of $\mathtt{e}$.

The problems above are attributed to the rules $F\,x \to \mathtt{e}$ and $F\,x \to x$, which respectively allow F to ignore arguments and to behave like an identity function. This section formalizes a type-based transformation that can remove such "non-productive" behaviors of non-terminals. A complication arises because (i) the grammars must actually be extended to enable such transformation, and (ii) the kinds of non-productive behaviors that should be removed depends on the order of grammars (more need to be eliminated with the increase of the order) and we have not yet obtained a general characterization of non-productive behaviors. We thus first present extended grammars in Section 3.1, and formalize the transformation by parametrizing it with a set of prohibited behaviors in Section 3.2. In the next section, we provide a sufficient characterization of prohibited behaviors for the order-2 case, and show that the removal of those behaviors indeed guarantee that the size of intermediate terms is linearly bounded by a generated tree.

3.1 Extended Grammars

This section introduces extended grammars, which are used as the target of the transformation.

Definition 4 (extended terms). *The set of* ***extended terms****, ranged over by* e*, is defined by:*

$$e ::= a \mid x \mid A \mid e\,E \mid \langle f \rangle\, E \qquad E ::= \{e_1, \ldots, e_k\} \qquad f ::= e \mid \lambda x : \kappa.f$$

Here, A *ranges over non-terminals, and* $k > 0$ *in* $\{e_1, \ldots, e_k\}$*. We require that* f *in* $\langle f \rangle$ *contains no non-terminals, terminals, nor free variables.*

Intuitively, $e\,\{e_1, \ldots, e_k\}$ applies the function e to the argument $\{e_1, \ldots, e_k\}$, which non-deterministically evaluate to e_i for some i; however, e *must* use each $e_1, \ldots, e_k$ at least once. Thus, if we have a rule $A\,x \to \mathtt{a}\,x\,x$, then $A\,\{e_1, e_2\}$ may be reduced to $\mathtt{a}\,e_1\,e_2$ or $\mathtt{a}\,e_2\,e_1$ but not to $\mathtt{a}\,e_1\,e_1$. We often write $e\,e_1$ for $e\,\{e_1\}$. The term $\langle f \rangle$ is semantically the same as the (extended) λ-term f. Note that $\langle f \rangle$ cannot occur in an argument position; for example, $A\,\langle \lambda x.x \rangle$ is disallowed. (To save the number of rules, however, we allow e to be instantiated to $\langle f \rangle$ in the definitions of the type judgment and substitutions below.) We later restrict the set of terms f that may occur in the form of $\langle f \rangle$.

The type judgment relation $\mathcal{K} \vdash_E e : \kappa$ is defined inductively by:

$$\frac{}{\{x : \kappa\} \vdash_E x : \kappa} \qquad \frac{}{\{A : \kappa\} \vdash_E A : \kappa} \qquad \frac{}{\emptyset \vdash_E a : \underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{\Sigma(a)} \to \mathtt{o}}$$

$$\frac{\{x_1 : \kappa_1, \ldots, x_k : \kappa_k\} \vdash_E e : \mathtt{o}}{\vdash_E \langle \lambda x_1 : \kappa_1. \cdots \lambda x_k : \kappa_k.e \rangle : \kappa_1 \to \cdots \to \kappa_k \to \mathtt{o}}$$

$$\frac{\mathcal{K}_1 \vdash_E e_1 : \kappa_2 \to \kappa \qquad \mathcal{K}_2 \vdash_E E_2 : \kappa_2}{\mathcal{K}_1 \cup \mathcal{K}_2 \vdash_E e_1\,E_2 : \kappa} \qquad \frac{\mathcal{K}_i \vdash_E e_i : \kappa \text{ for each } i \in I}{\bigcup_{i \in I} \mathcal{K}_i \vdash_E \{e_i \mid i \in I\} : \kappa}$$

Please notice that weakening is not allowed in the above rules. Therefore, if $\mathcal{K} \vdash_E e : \kappa$, then every variable in $\mathcal{K}$ must occur at least once in e.

Definition 5 (extended grammars). *A* ***combinator*** *is an extended* λ*-term* f *such that* $\emptyset \vdash_E f : \kappa$ *for some* κ*. Let* $\mathcal{C}$ *be a finite set of combinators. An* ***extended grammar*** *over* $\mathcal{C}$ *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* ***extended rewriting rules*** *of the form* $A\,x_1 \cdots x_\ell \to e$*, where* $A \in dom(\mathcal{N})$*, and* $f \in \mathcal{C}$ *for every* $\langle f \rangle$ *in* e*. We require that* $\mathcal{N}(A)$ *must be of the form* $\kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o}$ *and* $\Gamma \cup \{x_1 : \kappa_1, \ldots, x_\ell : \kappa_\ell\} \vdash_E e : \mathtt{o}$ *must hold for some* $\Gamma \subseteq \mathcal{N}$*. Furthermore,* $\lambda x_1. \cdots \lambda x_\ell.e \notin \mathcal{C}$*, and* e *must not contain a subterm of the form* $\langle \lambda x_1 \cdots x_k.e' \rangle\, E_1 \cdots E_k$*. (iv)* S *is a non-terminal called* ***the start symbol****, and* $\mathcal{N}(S) = \mathtt{o}$*. As before, the order and arity of* $\mathcal{G}$*, written* $\mathtt{order}(\mathcal{G})$ *and* $\mathtt{ar}(\mathcal{G})$*, are the largest order and arity of the types of non-terminals.*

To define the rewriting relation for extended grammars, we need to extend the ordinary notion of substitutions. An (extended) **substitution** is a map from variables to sets of terms. We write $[E_1/x_1, \ldots, E_k/x_k]$ for the substitution that

maps x_i to E_i, and use the meta-variable θ. The operation $[E/x]e$ replaces each occurrence of x in e with an element of E in a non-deterministic manner. Thus, we define the substitution operation as a relation $\theta \models e \rightsquigarrow e'$, which means that e' is the term obtained by applying the substitution θ to e. The relations $\theta \models e \rightsquigarrow e'$ and $\theta \models E \rightsquigarrow E'$ are defined inductively by:

$$\frac{}{[\,] \models a \rightsquigarrow a} \qquad \frac{}{[\,] \models A \rightsquigarrow A} \qquad \frac{}{[\,] \models \langle f \rangle \rightsquigarrow \langle f \rangle} \qquad \frac{}{[\{e\}/x] \models x \rightsquigarrow e}$$

$$\frac{\theta_1 \models e_1 \rightsquigarrow e_1' \quad \theta_2 \models E_2 \rightsquigarrow E_2'}{\theta_1 \cup \theta_2 \models e_1\, E_2 \rightsquigarrow e_1'\, E_2'} \qquad \frac{\theta_{i,j} \models e_i \rightsquigarrow e_{i,j} \text{ for each } i \in I, j \in J_i}{\bigcup_{i \in I, j \in J_i} \theta_{i,j} \models \{e_i \mid i \in I\} \rightsquigarrow \{e_{i,j} \mid i \in I, j \in J_i\}}$$

Here, the operation $\theta_0 \cup \theta_1$ on substitutions is defined by: (i) $dom(\theta_0 \cup \theta_1) = dom(\theta_0) \cup dom(\theta_1)$; (ii) $(\theta_0 \cup \theta_1)(x) = \theta_0(x) \cup \theta_1(x)$ if $x \in dom(\theta_0) \cap dom(\theta_1)$, and (iii) $(\theta_0 \cup \theta_1)(x) = \theta_i(x)$ if $x \in dom(\theta_i) \setminus dom(\theta_{1-i})$.

Example 3. Let $\theta = [\{\mathtt{b}, \mathtt{c}\}/x]$ and $e = \mathtt{a}\, x\, x$. Then $\theta \models e \rightsquigarrow \mathtt{a\,b\,c}$ and $\theta \models e \rightsquigarrow \mathtt{a\,c\,b}$ hold, but neither $\theta \models e \rightsquigarrow \mathtt{a\,b\,b}$ nor $\theta \models e \rightsquigarrow \mathtt{a\,c\,c}$ does.

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

$$\frac{A\, x_1 \cdots x_k \to e \in \mathcal{R} \quad [E_1/x_1, \ldots, E_k/x_k] \models e \rightsquigarrow e'}{A\, E_1 \cdots E_k \longrightarrow_{\mathcal{G}} e'} \text{ (ER-NT)} \qquad \frac{[E_1/x_1, \ldots, E_k/x_k] \models e \rightsquigarrow e'}{\langle \lambda x_1 \cdots x_k.e \rangle\, E_1 \cdots E_k \longrightarrow_{\mathcal{G}} e'} \text{ (ER-Comb)}$$

$$\frac{e_i \longrightarrow_{\mathcal{G}} e_i' \qquad i \in \{1, \ldots, \Sigma(a)\}}{a\,\{e_1\} \cdots \{e_{\Sigma(a)}\} \longrightarrow_{\mathcal{G}} a\,\{e_1\} \cdots \{e_{i-1}\}\,\{e_i'\}\,\{e_{i+1}\} \cdots \{e_{\Sigma(a)}\}} \text{ (ER-Cong)}$$

We often omit the subscript $\mathcal{G}$. The **tree language generated by** an extended grammar $\mathcal{G}$, written $\mathcal{L}(\mathcal{G})$, is the set $\{\pi \in \mathbf{Tree}_{\Sigma_{\mathcal{G}}} \mid S \longrightarrow_{\mathcal{G}}^{*} \pi\}$ (where we identify a singleton set $\{e\}$ with e; for example, the extended term $\mathtt{a}\,\{\mathtt{e}\}\,\{\mathtt{e}\}$ is interpreted as the tree $\mathtt{a\,e\,e}$).

Example 4. Consider the extended grammar $\mathcal{G}_2 = (\{\mathtt{a}{:}2, \mathtt{b}{:}0, \mathtt{c}{:}0\}, \{S{:}\mathtt{o}, F{:}\mathtt{o} \to \mathtt{o}\}, \mathcal{R}, S)$ where $\mathcal{R} = \{S \to F\,\{\mathtt{b}, \mathtt{c}\}, F\,x \to \mathtt{a}\{F\,\{x\}\}\{F\,\{x\}\}, F\,x \to x\}$, then:

$$S \longrightarrow F\{\mathtt{b}, \mathtt{c}\} \longrightarrow \mathtt{a}(F\{\mathtt{b}\})\,(F\{\mathtt{b}, \mathtt{c}\}) \longrightarrow^{*} \mathtt{a\,b}\,(\mathtt{a}\,(F\{\mathtt{c}\})\,(F\{\mathtt{b}\})) \longrightarrow^{*} \mathtt{a\,b}\,(\mathtt{a\,c\,b}).$$

$\mathcal{L}(\mathcal{G}_2)$ is the set of all binary trees that contain at least one $\mathtt{b}$ and one $\mathtt{c}$.

Reduction with Eager Normalization. We define $e \longrightarrow_{\lambda} e'$ inductively by: (i) $e \longrightarrow_{\lambda} e'$ if $e \longrightarrow_{\mathcal{G}} e'$ is derivable by using rule ER-Comb, (ii) $eE \longrightarrow_{\lambda} e'E$ if $e \longrightarrow_{\lambda} e'$; (iii) $e_0\,(E \uplus \{e\}) \longrightarrow_{\lambda} e_0\,(E \cup \{e_1, \ldots, e_k\})$ if $e \longrightarrow_{\lambda} e_i$ for each $i \in \{1, \ldots, k\}$ with $k \geq 1$; (iv) $(\lambda x.e)E \longrightarrow_{\lambda} e'$ if $[E/x] \models e \rightsquigarrow e'$; (v) $\lambda x.e \longrightarrow_{\lambda} \lambda x.e'$ if $e \longrightarrow_{\lambda} e'$; and (vi) $\langle f_1 \rangle(\langle f_2 \rangle\, E) \longrightarrow_{\lambda} \langle f \rangle E$ if $\lambda x.f_1(f_2\, x) \longrightarrow_{\lambda} f \in \mathcal{C}$. In the above definition, we have extended the syntax of extended terms and allowed λ-abstractions to occur outside $\langle \cdot \rangle$, but ordinary extended terms are closed under $\longrightarrow_{\lambda}$. In $e \longrightarrow_{\lambda} e'$, we implicitly require that every argument of a terminal symbol must be a singleton set both in e and e'.

Henceforth, we assume that the set $\mathcal{C}$ is closed under composition, in the sense that if $f_1, f_2 \in \mathcal{C}$ and $\lambda x.f_1(f_2\,x) \longrightarrow_\lambda^* e$, then $e \longrightarrow_\lambda^* f$ for some $f \in \mathcal{C}$. We write $e \downarrow_\lambda e'$ if $e \longrightarrow_\lambda^* e' \not\longrightarrow_\lambda$, and write $e \Longrightarrow_{\mathcal{G}} e'$ if $e(\downarrow_\lambda \cdot \longrightarrow_{\mathcal{G}} \cdot \downarrow_\lambda)e'$. For every term e of type o and tree π, $e \longrightarrow_{\mathcal{G}}^* \pi$ if and only if $e \Longrightarrow_{\mathcal{G}}^* \pi$. In Section 4, we bound the size of intermediate terms in a rewriting sequence $S \Longrightarrow_{\mathcal{G}}^* \pi$.

3.2 From Grammars to Extended Grammars

This section presents a translation from (ordinary) grammars to extended grammars over a finite set $\mathcal{C}$ of combinators, and shows that the translation preserves the tree language. We use type-based transformation techniques to eliminate useless arguments and (non-applied) combinators in $\mathcal{C}$.

Definition 6 (intersection types). *The set of* ***intersection types*** *over $\mathcal{C}$, ranged over by τ, is given by:*

$$\tau ::= \mathsf{o} \mid (\sigma_1 \to \cdots \to \sigma_k \to \mathsf{o}, \eta) \quad \sigma ::= \bigwedge\{\tau_1, \ldots, \tau_\ell\} \quad \eta \ \textit{(flag)} \ ::= \mathbf{nc} \mid \langle f \rangle$$

Here, f ranges over $\mathcal{C}$. We define $\mathit{flag}(\tau)$ by $\mathit{flag}(\mathsf{o}) = \mathbf{nc}$ and $\mathit{flag}(\sigma_1 \to \cdots \to \sigma_k \to \mathsf{o}, \eta) = \eta$.

We often write $\tau_1 \wedge \cdots \wedge \tau_k$ and $\top$ for $\bigwedge\{\tau_1, \ldots, \tau_k\}$ and $\bigwedge \emptyset$ respectively. We assume a certain total order $<$ on the intersection types. Intuitively, the type o describes trees. The type $\bigwedge\{\tau_1, \ldots, \tau_\ell\}$ describes terms that behave like a value of type τ_i for every $i \in \{1, \ldots, \ell\}$. The type $(\sigma_1 \to \cdots \to \sigma_k \to \mathsf{o}, \eta)$ describes functions that take arguments of types $\sigma_1, \ldots, \sigma_k$ and return a tree of type o. The flag η describes how the term behaves *after the transformation* for removing unused arguments. If $\eta = \langle f \rangle$, then the term behaves like f after the transformation, and if $\eta = \mathbf{nc}$, the term does not behave like any of the combinators in $\mathcal{C}$. For example, the term $\lambda x.\lambda y.y$ has type $(\top \to \mathsf{o} \to \mathsf{o}, \langle \lambda y.y \rangle)$, because after removing the redundant argument x, the term behaves like the identity function $\lambda y.y$.

We consider only types that respect underlying sorts. The operation $[\![\cdot]\!]$ given below maps an intersection type to the simple type obtained by the grammar transformation.

$$[\![(\widetilde{\sigma} \to \mathsf{o}, \eta)]\!] = [\![\widetilde{\sigma} \to \mathsf{o}\,]\!] \qquad [\![\mathsf{o}]\!] = \mathsf{o}$$
$$[\![\textstyle\bigwedge\{\tau_1, \ldots, \tau_\ell, \tau'_1, \ldots, \tau'_{\ell'}\} \to \widetilde{\sigma} \to \mathsf{o}]\!] = [\![\tau_1]\!] \to \cdots \to [\![\tau_\ell]\!] \to [\![\widetilde{\sigma} \to \mathsf{o}]\!]$$
$$\text{if } \mathit{flag}(\tau'_j) \neq \mathbf{nc} \text{ and } \mathit{flag}(\tau_j) = \mathbf{nc} \text{ and } j < j' \text{ implies } \tau_j < \tau_{j'}$$

Here, $\widetilde{\sigma} \to \mathsf{o}$ is an abbreviation of $\sigma_1 \to \cdots \to \sigma_k \to \mathsf{o}$. The type τ is called a **refinement** of κ, if $\tau :: \kappa$ is derivable by the following rules.

$$\frac{}{\mathsf{o} :: \mathsf{o}} \qquad \frac{\sigma_i :: \kappa_i \text{ for each } i \in \{1, \ldots, k\} \qquad \emptyset \vdash \langle f \rangle : [\![(\widetilde{\sigma} \to \mathsf{o}, f)]\!]}{(\widetilde{\sigma} \to \mathsf{o}, \langle f \rangle) :: \widetilde{\kappa} \to \mathsf{o}}$$

$$\frac{\tau_i :: \kappa \text{ for each } i \in \{1, \ldots, k\}}{\bigwedge\{\tau_1, \ldots, \tau_k\} :: \kappa} \qquad \frac{\sigma_i :: \kappa_i \text{ for each } i \in \{1, \ldots, k\}}{(\widetilde{\sigma} \to \mathsf{o}, \mathbf{nc}) :: \widetilde{\kappa} \to \mathsf{o}}$$

Henceforth we consider only intersection types that are refinement of some simple types. For example, intersection types like $\bigwedge\{\mathsf{o}, (\mathsf{o} \to \mathsf{o}, \mathbf{nc})\} \to \mathsf{o}$ and $(\mathsf{o} \to \mathsf{o}, \langle \lambda f.\lambda x.f(x) \rangle)$ are excluded out.

Transformation rules. We define the term transformation relation $\Gamma \vdash t : \tau \Rightarrow e$, where: (i) Γ is an (intersection) type environment, i.e., a set of type bindings of the form $\{x_1 : \tau_1, \ldots, x_k : \tau_k\}$, where each variable may occur more than once (we often omit curly brackets and just write $x_1 : \tau_1, \ldots, x_k : \tau_k$); (ii) t is a term; (iii) τ is the type of t; and (iv) e is an extended term. When $\sigma = \bigwedge\{\tau_1, \ldots, \tau_k\}$, we sometimes write $x : \sigma$ for $x : \tau_1, \ldots, x : \tau_k$. Intuitively, $\Gamma \vdash t : \tau \Rightarrow e$ means that the term t corresponds to e, when t behaves as specified by τ. For example, if $\Gamma = \{g : (\mathtt{o} \to \mathtt{o}, \langle \lambda x.x \rangle)\}$, then $\Gamma \vdash g\,\mathtt{e} : \tau \Rightarrow \langle \lambda x.x \rangle \mathtt{e}$ should hold, since Γ says that g will be transformed to a term that behaves like $\lambda x.x$.

The transformation relation is inductively defined by the following rules:

$$\frac{\mathit{flag}(\tau) = \langle f \rangle}{x : \tau \vdash x : \tau \Rightarrow \langle f \rangle} \quad \text{(X-VarC)} \qquad \frac{\mathit{flag}(\tau) = \mathbf{nc}}{x : \tau \vdash x : \tau \Rightarrow x_\tau} \quad \text{(X-Var)}$$

$$\frac{}{\emptyset \vdash a : (\underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{\Sigma(a)} \to \mathtt{o}, \mathbf{nc}) \Rightarrow a} \quad \text{(X-T)} \qquad \frac{\mathit{flag}(\tau) = \mathbf{nc}}{\emptyset \vdash A : \tau \Rightarrow A_\tau} \quad \text{(X-NT)}$$

$$\frac{\begin{array}{c} A\,x_1 \cdots x_k \to t \in \mathcal{R} \qquad f = \lambda\mathbf{Vars}(\{x_1 : \sigma_1, \ldots, x_k : \sigma_k\}, x_1 \cdots x_k).e \in \mathcal{C} \\ \tau = (\sigma_1 \to \cdots \to \sigma_k \to \mathtt{o}, \langle f \rangle) \qquad x_1 : \sigma_1, \ldots, x_k : \sigma_k \vdash t : \mathtt{o} \Rightarrow e \end{array}}{\emptyset \vdash A : \tau \Rightarrow \langle f \rangle} \quad \text{(X-NTC)}$$

$$\frac{\begin{array}{c} \Gamma_0 \vdash t_0 : (\bigwedge\{\tau_1, \ldots, \tau_\ell\} \to \rho, \eta) \Rightarrow e_0 \qquad \eta' = \begin{cases} \eta & \text{if } k = 0 \\ \mathbf{nc} & \text{if } k > 0 \end{cases} \\ \Gamma_i \vdash t_1 : \tau_i \Rightarrow E_i \qquad \mathit{flag}(\tau_i) = \mathbf{nc} \text{ for } i \in \{1, \ldots, k\} \qquad \tau_i < \tau_j \text{ if } i < j \leq k \\ \Gamma_i \vdash t_1 : \tau_i \Rightarrow e_{1,i} \qquad \mathit{flag}(\tau_i) \neq \mathbf{nc} \text{ for } i \in \{k+1, \ldots, \ell\} \end{array}}{\Gamma_0 \cup \bigcup_{i \in \{1,\ldots,\ell\}} \Gamma_i \vdash t_0 t_1 : (\rho, \eta') \Rightarrow e_0\,E_1 \cdots E_k} \quad \text{(X-App)}$$

$$\frac{\Gamma \vdash t : \tau \Rightarrow \langle \lambda x_1. \cdots \lambda x_k.e_0 \rangle\,E_1 \cdots E_k \qquad [E_1/x_1, \ldots, E_k/x_k] \models e_0 \rightsquigarrow e}{\Gamma \vdash t : \tau \Rightarrow e} \quad \text{(X-Red)}$$

$$\frac{\Gamma_i \vdash t : \tau \Rightarrow e_i \text{ for each } i \in \{1, \ldots, k\} \qquad k \geq 1}{\Gamma_1 \cup \cdots \cup \Gamma_k \vdash t : \tau \Rightarrow \{e_1, \ldots, e_k\}} \quad \text{(X-Set)}$$

In the rule X-NTC above, $\mathbf{Vars}(\Gamma, \widetilde{x})$ (where $\widetilde{x}$ is a possibly empty sequence of variables) is a sequence of type bindings defined by (recall that $<$ is the total order on intersection types):

$$\mathbf{Vars}(\Gamma, \epsilon) = \epsilon \qquad \mathbf{Vars}(\Gamma, x\widetilde{y}) = (x_{\tau_1} : [\![\tau_1]\!]) \cdots (x_{\tau_k} : [\![\tau_k]\!])\mathbf{Vars}(\Gamma, \widetilde{y})$$
$$\text{where } \{\tau_1, \ldots, \tau_k\} = \{\tau \mid x : \tau \in \Gamma, \mathit{flag}(\tau) = \mathbf{nc}\} \text{ and } \tau_1 < \cdots < \tau_k.$$

Here is some explanation of the transformation rules. The rule X-VarC ensures that if x behaves like f, then x is replaced with $\langle f \rangle$; this allows us to propagate information about elements of $\mathcal{C}$ during the transformation, and avoid

passing them around as function arguments. The rule X-VAR says that if x does not behave like an element of $\mathcal{C}$, then the variable is replicated for each type τ. (Here, we assume that x_τ and $x'_{\tau'}$ are different variables if $x \neq x'$ or $\tau \neq \tau'$.) Similarly, there are two rules for non-terminals, depending on whether the body of a rule behaves like an element of $\mathcal{C}$. The rule X-APP is for applications. We ensure that only terms with **nc** flags remain as arguments, so that terms behaving like elements of $\mathcal{C}$ are not passed around. Each argument is now a *set* of terms; this is because the output of transformation may not be unique. For example, if F has both types $(\mathtt{o} \to \top \to \mathtt{o}, \mathbf{nc})$ and $(\top \to \mathtt{o} \to \mathtt{o}, \mathbf{nc})$ (which means that F may use either the first or second argument), then $F\,\mathtt{b}\,\mathtt{c}$ in an argument position would be replaced by $\{F_{(\mathtt{o}\to\top\to\mathtt{o},\mathbf{nc})}\,\mathtt{b},\ F_{(\top\to\mathtt{o}\to\mathtt{o},\mathbf{nc})}\,\mathtt{c}\}$.

For a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an extended one $\mathcal{G}' = (\Sigma, \mathcal{N}', \mathcal{R}', S_\mathtt{o})$, we write $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$ if (i) $\mathcal{N}' = \{F_\tau \mapsto [\![\tau]\!] \mid \tau :: \mathcal{N}(F)\}$ and (ii) $\mathcal{R}'$ is the set:

$$\begin{array}{l}\{F_{(\sigma_1\to\cdots\to\sigma_k\to\mathtt{o},\mathbf{nc})}\,y_1\ \cdots\ y_m \to e \mid \\ \quad (F\,x_1\ \cdots\ x_k \to t) \in \mathcal{R} \wedge x_1 : \sigma_1, \ldots, x_k : \sigma_k \vdash t : \mathtt{o} \Rightarrow e \\ \quad \wedge \mathbf{Vars}(\{x_1 : \sigma_1, \ldots, x_k : \sigma_k\}, x_1 \cdots x_k) = (y_1 : \kappa_1) \cdots (y_m : \kappa_m) \\ \quad \wedge (\sigma_1 \to \cdots \to \sigma_k \to \mathtt{o}, \mathbf{nc}) :: \mathcal{N}(F) \wedge \lambda y_1 : \kappa_1. \cdots \lambda y_m : \kappa_m. e \notin \mathcal{C} \wedge e \not\longrightarrow_\lambda\}.\end{array}$$

So far we have implicitly assumed the set $\mathcal{C}$ is fixed when we write $\Gamma \vdash t : \tau \Rightarrow e$ and $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$. We write $\Gamma \vdash_\mathcal{C} t : \tau \Rightarrow e$ and $\vdash_\mathcal{C} \mathcal{G} \Rightarrow \mathcal{G}'$ if we wish to make the set $\mathcal{C}$ explicit.

Example 5. Recall $\mathcal{G}_0$ in Example 1. Let $\mathcal{C} = \{\lambda g.\lambda x.g\,x, \lambda g.\lambda x.g\,x\,x\}$. By applying the transformation and removing redundant rules, we obtain the grammar $\mathcal{G}'_0 = (\Sigma, \mathcal{N}', \mathcal{R}', S_\mathtt{o})$, where $f = \lambda g.\lambda x.g\,x\,x$ and $\tau = ((\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \to \mathtt{o} \to \mathtt{o}, \mathbf{nc})$ with:

$$\begin{array}{l}\mathcal{N}' = \{S_\mathtt{o} : \mathtt{o}, F_\tau : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o}, T_\tau : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o}\} \\ \mathcal{R}' = \{S_\mathtt{o} \to \mathtt{a}\,\mathtt{e}\,\mathtt{e}, \quad S_\mathtt{o} \to \mathtt{b}\,\mathtt{e}\,\mathtt{e}, \quad S_\mathtt{o} \to F_\tau\,\{\langle f\rangle\,\mathtt{a}\}\,\mathtt{e}, \\ \qquad S_\mathtt{o} \to F_\tau\,\{\langle f\rangle\,\mathtt{b}\}\,\mathtt{e}, \quad S_\mathtt{o} \to F_\tau\,\{\langle f\rangle\,\mathtt{a}, \langle f\rangle\,\mathtt{b}\}\,\mathtt{e}, \\ \qquad F_\tau\,g\,x \to T_\tau\,g\,x, \quad F_\tau\,g\,x \to F_\tau(T_\tau\,g)\,x, \quad T_\tau\,g\,x \to g(g\,x)\}.\end{array}$$

The tree $\mathtt{a}\,(\mathtt{b}\,\mathtt{e}\,\mathtt{e})\,(\mathtt{a}\,\mathtt{e}\,\mathtt{e})$ is obtained as follows. (We omit the subscripts of non-terminals, as they happen to be the same for each original non-terminal.)

$$\begin{array}{l}S \longrightarrow F\,\{\langle f\rangle\,\mathtt{a}, \langle f\rangle\,\mathtt{b}\}\,\mathtt{e} \longrightarrow T\,\{\langle f\rangle\,\mathtt{a}, \langle f\rangle\,\mathtt{b}\}\,\mathtt{e} \longrightarrow \langle f\rangle\,\mathtt{a}\{\langle f\rangle\,\mathtt{a}\,\mathtt{e}, \langle f\rangle\,\mathtt{b}\,\mathtt{e}\} \\ \longrightarrow \mathtt{a}\,(\langle f\rangle\,\mathtt{b}\,\mathtt{e})\,(\langle f\rangle\,\mathtt{a}\,\mathtt{e}) \longrightarrow^* \mathtt{a}\,(\mathtt{b}\,\mathtt{e}\,\mathtt{e})\,(\mathtt{a}\,\mathtt{e}\,\mathtt{e}).\end{array}$$

The following theorem states that the transformation preserves the language.

Theorem 1. *If $\mathcal{G}$ is an order-n grammar and $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$, then $\mathcal{G}'$ is a valid order-n extended grammar and $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.*

4 Bounding the Size of Intermediate Terms

In this section, we restrict the order of grammars to 2, and let $\mathcal{C}$ be the following set $\mathcal{C}_m$:

$$\begin{array}{l}\{\lambda x : \mathtt{o}.x\} \cup \\ \{\lambda y_1 \cdots y_k.y_i\,E_1\ \cdots\ E_\ell \mid k, \ell \leq m, \text{ and } E_1 \cup \cdots \cup E_\ell = \{y_1, \ldots, y_k\} \setminus \{y_i\}\}.\end{array}$$

We shall show that for an extended order-2 grammar over $\mathcal{C}$, the size of intermediate terms occurring in a production of a tree π is linearly bounded by the size of π. The **size** $|e|$ of an extended term e is defined by:

$$|a| = |x| = |A| = 1$$
$$|e\,\{e_1, \ldots, e_k\}| = |e| + |e_1| + \cdots + |e_k| \quad |\langle f \rangle\,\{e_1, \ldots, e_k\}| = 1 + |e_1| + \cdots + |e_k|.$$

Here, $e_1, \ldots, e_k$ are different from each other. The size $|\pi|$ of a tree π is the size of π as an extended term, which is the same as the number of nodes and leaves of π. The property mentioned above is stated more formally as follows.

Theorem 2. *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be an order-2 extended grammar over $\mathcal{C}_m$ with $\mathtt{ar}(\mathcal{G}) \leq m$. Then there exists an (effectively computable) constant c such that for every tree $\pi \in \mathbf{Tree}_\Sigma$, if $S_\mathtt{o} \Longrightarrow^*_{\mathcal{G}'} e \Longrightarrow^*_{\mathcal{G}'} \pi$, then $|e| \leq c|\pi|$.*

The following main result of this paper is obtained as a corollary:

Corollary 1. *Fix an order-2 grammar $\mathcal{G}$, Then the membership problem $\pi \stackrel{?}{\in} \mathcal{L}(\mathcal{G})$ can be decided in a non-deterministic Turing machine in $O(|\pi|)$ space.*

Proof. Suppose $\mathtt{ar}(\mathcal{G}) = k'$ and $k = \max(k', 2)$. We first determine m of $\mathcal{C}_m$. For each order-1 type κ of arity k, the number of intersection types such that $\tau :: \kappa$ and $\mathit{flag}(\tau) = \mathbf{nc}$ is 2^k. Thus, for each order-2 type $\kappa = \kappa_1 \to \cdots \to \kappa_j \to \mathtt{o}$ (with $j \leq k$), the arity of $[\![\sigma]\!]$ for σ such that $\sigma :: \kappa$ is at most $k \times 2^k$. Let $m = k \times 2^k$ and $\mathcal{C} = \mathcal{C}_m$. By Theorem 1, we can effectively construct an order-2 extended grammar $\mathcal{G}'$ over $\mathcal{C}_m$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$. By the above reasoning, $\mathtt{ar}(\mathcal{G}') \leq m$. Compute the constant c of Theorem 2. Since $\mathcal{G}$ is fixed, those steps can be performed offline. Given π, one can non-deterministically apply reductions by $\Longrightarrow_{\mathcal{G}}$ either until π is obtained (and answer yes only in this case), the size of a term exceeds $c|\pi|$, or the reduction gets stuck. By Theorem 2, there is an execution sequence that outputs yes if and only if $\pi \in \mathcal{L}(\mathcal{G}')$. Since $\mathcal{G}$ is fixed (therefore non-terminals, terminals, and $\mathcal{C}_m$ are also fixed), the actual space required for storing each intermediate term e is also linearly bounded by $|e| \leq c|\pi|$; hence this computation can be simulated by a non-deterministic Turing machine with $O(|\pi|)$ space. □

We sketch the proof of Theorem 2 in the rest of this section. We call a λ-term of the form $\lambda x_1.\cdots\lambda x_k.x_{\theta(1)}x_{\theta(2)}\cdots x_{\theta(k)}$ (where $k \geq 1$ and θ is a permutation on $\{1, \ldots, k\}$) an **extended permutator**. The proof consists of two steps. In the first step, from the reduction sequence $e \Longrightarrow^*_{\mathcal{G}} \pi$, we construct a term M of the linear λ-calculus that simulates the behavior of e in $e \Longrightarrow^*_{\mathcal{G}} \pi$, such that $|e|$ is bounded by the measure $\mathit{asize}(M)$ defined below (which is the number of top-level abstractions and variables), and M contains extended permutators only in restricted positions. In the second step, we show that any linear λ-term M that satisfies the conditions above is linearly bounded by $|\pi|$. For space restriction, we discuss only the first step below. Details about the first step and the second step are found in the extended version.

We first define a translation from extended grammars to linear λ-calculus with product types.

Definition 7. *The set of* ***linear types****, ranged over by* γ*, is given by:*

$$\gamma ::= \mathsf{o} \mid \gamma \times \cdots \times \gamma \to \gamma$$

We assume a total order $\leq$ *on linear types. The* ***refinement relation*** $\gamma :: \kappa$ *on types is defined by:*

$$\frac{}{\mathsf{o} :: \mathsf{o}} \qquad \frac{\gamma_{1,i} :: \kappa_1 \text{ for each } i \in \{1,\ldots,k\} \qquad \gamma_2 :: \kappa_2 \qquad \gamma_{1,1} \leq \cdots \leq \gamma_{1,k}}{(\gamma_{1,1} \times \cdots \times \gamma_{1,k} \to \gamma_2) :: (\kappa_1 \to \kappa_2)}$$

Henceforth we consider only types γ such that $\gamma :: \kappa$ for some κ.

Definition 8. *The set of* ***linear*** λ***-terms****, ranged over by* u*, is given by:*

$$u ::= x \mid uU \mid \lambda(x_1 : \gamma_1, \ldots, x_k : \gamma_k).u \qquad U ::= (u_1, \ldots, u_k)$$

A linear λ*-term* u *is called a* ***pure linear*** λ***-term*** *if the size of every tuple in* u *is* 1 *(i.e.,* $k = 1$ *for every subterm of the form* $\lambda(x_1, \ldots, x_k).u'$ *or* $(u_1, \ldots, u_k)$ *and every type* $\gamma_1 \times \cdots \times \gamma_k \to \gamma$*). We define* $asize(u)$ *by:*

$$asize(x) = asize(\lambda(x_1, \ldots, x_k).u) = 1$$
$$asize(u_0(u_1, \ldots, u_k)) = asize(u_0) + asize(u_1) + \cdots + asize(u_k).$$

We use a meta-variable M for pure linear λ-terms. We often omit parentheses for unary tuples, and write $\lambda x.u$ for $\lambda(x).u$, and u for (u).

The type judgment relation $\Delta \vdash_{\mathtt{L}} u : \gamma$ for linear λ-terms is given by:

$$\frac{}{\{x : \gamma\} \vdash_{\mathtt{L}} x : \gamma} \qquad \frac{\Delta \uplus \{x_1 : \gamma_1, \ldots, x_k : \gamma_k\} \vdash_{\mathtt{L}} u : \gamma}{\Delta \vdash_{\mathtt{L}} \lambda(x_1, \ldots, x_k).u : \gamma_1 \times \cdots \times \gamma_k \to \gamma}$$

$$\frac{\Delta_0 \vdash_{\mathtt{L}} u_0 : \gamma_1 \times \cdots \times \gamma_k \to \gamma \qquad \Delta_i \vdash_{\mathtt{L}} u_i : \gamma_i \text{ for each } i \in \{1,\ldots,k\}}{\Delta_0 \uplus \cdots \uplus \Delta_k \vdash_{\mathtt{L}} u_0(u_1, \ldots, u_k) : \gamma}$$

Here, $\Delta_0 \uplus \Delta_1$ is defined to be $\Delta_0 \cup \Delta_1$ only if $dom(\Delta_0) \cap dom(\Delta_1) = \emptyset$.

The transformation relations $\mathcal{K} \vdash e : \kappa \Rightarrow u : \gamma \dashv \Delta$ and $\mathcal{K} \vdash E : \kappa \Rightarrow U : \gamma_1 \times \cdots \times \gamma_k \dashv \Delta$ are defined by the rules below.

$$\frac{i \text{ fresh}}{\emptyset \vdash a : \underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{\Sigma(a)} \to \mathsf{o} \Rightarrow a^{(i)} : \underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{\Sigma(a)} \to \mathsf{o} \dashv a^{(i)} : \underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{\Sigma(a)} \to \mathsf{o}} \quad \text{(LX-Const)}$$

$$\frac{i \text{ fresh} \qquad \gamma :: \kappa}{\{x : \kappa\} \vdash x : \kappa \Rightarrow x^{(i)} : \gamma \dashv x^{(i)} : \gamma} \quad \text{(LX-V)} \qquad \frac{\emptyset \vdash f : \kappa \Rightarrow u : \gamma \dashv \emptyset}{\emptyset \vdash \langle f \rangle : \kappa \Rightarrow u : \gamma \dashv \emptyset} \quad \text{(LX-Com)}$$

$$\frac{\emptyset \vdash \lambda x_1.\cdots\lambda x_k.e : \kappa \Rightarrow u : \gamma \dashv \Delta \qquad F\, x_1\, \cdots\, x_k \to e \in \mathcal{R}}{\emptyset \vdash F : \kappa \Rightarrow u : \gamma \dashv \Delta} \quad \text{(LX-NT)}$$

$$\frac{\mathcal{K}_i \vdash e_i : \kappa \Rightarrow u_i : \gamma_i \dashv \Delta_i \text{ for each } i \in \{1,\dots,\ell\} \qquad \gamma_1 \le \cdots \le \gamma_\ell}{\mathcal{K}_1 \cup \cdots \cup \mathcal{K}_\ell \vdash \{e_1,\dots,e_\ell\} : \kappa \Rightarrow (u_1,\dots,u_\ell) : \gamma_1 \times \cdots \times \gamma_\ell \dashv \Delta_1 \uplus \cdots \uplus \Delta_\ell} \quad \text{(LX-Tset)}$$

$$\frac{\mathcal{K}_0 \vdash e_0 : \kappa_0 \to \kappa \Rightarrow u_0 : \gamma_1 \times \cdots \times \gamma_k \to \gamma \dashv \Delta_0 \qquad \mathcal{K}_1 \vdash E : \kappa_0 \Rightarrow U : \gamma_1 \times \cdots \times \gamma_k \dashv \Delta_1}{\mathcal{K}_0 \cup \mathcal{K}_1 \vdash e_0\, E : \kappa \Rightarrow u_0\, U : \gamma \dashv \Delta_0 \uplus \Delta_1} \quad \text{(LX-App)}$$

$$\frac{\mathcal{K} \cup \{x : \kappa_0\} \vdash e : \kappa \Rightarrow u : \gamma \dashv \Delta, x^{(i_1)} : \gamma_1, \dots, x^{(i_\ell)} : \gamma_\ell \qquad \gamma_1 \le \cdots \le \gamma_\ell \qquad x \notin dom(\mathcal{K})}{\mathcal{K} \vdash \lambda x.e : \kappa_0 \to \kappa \Rightarrow \lambda(x^{(i_1)},\dots,x^{(i_\ell)}).u : \gamma_1 \times \cdots \times \gamma_\ell \to \gamma \dashv \Delta} \quad \text{(LX-Ab)}$$

The idea is to replicate each variable and terminal for each use in a rewriting sequence $e \longrightarrow^*_{\mathcal{G}} \pi$. In rule LX-Const, $a^{(i)}$ obtained by the translation is treated as a variable. In LX-NT, a non-terminal is (non-deterministically) expanded, and then transformed to a linear λ-term. In LX-Tset, we allow $e_i = e_j$ even if $i \neq j$.

Example 6. Recall $\mathcal{G}'_0$ in Example 5. The term $T\,\{\langle f\rangle \mathtt{a}, \langle f\rangle \mathtt{b}\}\,\{\mathtt{e}\}$ occurring in the production of $\mathtt{a}\,(\mathtt{b}\,\mathtt{e}\,\mathtt{e})\,(\mathtt{a}\,\mathtt{e}\,\mathtt{e})$ is transformed to:

$$\begin{array}{l}\Big(\lambda(g^{(1)},g^{(2)},g^{(3)}).\lambda(x^{(1)},x^{(2)},x^{(3)},x^{(4)}).g^{(1)}\big(g^{(2)}(x^{(1)},x^{(2)}),(g^{(3)}(x^{(3)},x^{(4)}))\big)\Big)\\ \Big(\big(\lambda g.\lambda(y^{(1)},y^{(2)}).g(y^{(1)},y^{(2)})\big)\mathtt{a}^{(1)},\big(\lambda g.\lambda(y^{(1)},y^{(2)}).g(y^{(1)},y^{(2)})\big)\mathtt{b}^{(2)},\\ \quad\big(\lambda g.\lambda(y^{(1)},y^{(2)}).g(y^{(1)},y^{(2)})\big)\mathtt{a}^{(3)}\Big)\\ (\mathtt{e}^{(1)},\mathtt{e}^{(2)},\mathtt{e}^{(3)},\mathtt{e}^{(4)})\end{array}$$

with $\Delta = \mathtt{a}^{(1)} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{b}^{(2)} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{a}^{(3)} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{e}^{(1)} : \mathtt{o}, \mathtt{e}^{(2)} : \mathtt{o}, \mathtt{e}^{(3)} : \mathtt{o}, \mathtt{e}^{(4)} : \mathtt{o}$. Here we have reused labels (for i in LX-V) when there is no danger of variable confusion.

The transformation satisfies the following property.

Theorem 3. *If $e \longrightarrow^*_{\mathcal{G}} \pi$, then there exists u such that $\emptyset \vdash e : \mathtt{o} \Rightarrow u : \mathtt{o} \dashv \Delta$ where for each terminal symbol a, the number of bindings of the form $a^{(i)}$ in Δ is the same as the number of occurrences of a in π.*

We can obtain the following property from the above theorem.

Theorem 4. *Let $\mathcal{G}$ be an order-2 extended grammar over $\mathcal{C}_m$ with $\mathtt{ar}(\mathcal{G}) \le m$. If $S \Longrightarrow^*_{\mathcal{G}} e \Longrightarrow^*_{\mathcal{G}} \pi$, then there exists a pure linear λ-term M that satisfies: (i) $\Delta \vdash_{\mathtt{L}} M : \mathtt{o}$; (ii) $codom(\Delta) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$ and $|\{x \mid \Delta(x) = \mathtt{o}\}|$ equals the number of leaves of π; (iii) $asize(M) \ge |e|$; (iv) M contains only top-level β-redexes; and (v) M does not contain any extended permutator in an argument position, nor any consecutive application of extended permutators.*

Proof Sketch. Since $e \Longrightarrow^*_{\mathcal{G}} \pi$, we also have $e \longrightarrow^*_{\mathcal{G}} \pi$. Thus, one can construct a term u that satisfies the condition of Theorem 3. Let M be the pure linear λ-term obtained from u by applying the currying transformation, and then normalizing all the redexes under λ-abstraction. Then M satisfies the required conditions. $\square$

In the second step, we show that $\mathit{asize}(M) \leq 28|\{x \mid x : \mathsf{o} \in \Delta\}|$ holds for any pure linear λ-term M and type environment Δ that satisfy the conditions (i), (ii), (iv), and (v), from which Theorem 2 follows.

5 Related Work

As mentioned in Section 1, higher-order (formal) languages have been introduced in 1970's and actively studied since then, but a number of problems remain open especially about *unsafe* higher-order languages. Inaba and Maneth [6] proved that any *safe* higher-order (word) languages are context-sensitive; they actually proved the stronger result that the membership is in the intersection of *deterministic* linear space and NP. Context-sensitiveness of *unsafe* higher-order languages has been open (for order-2 or higher for the tree language case, and for order-3 or higher for the word language case).

Type-based techniques for reasoning about higher-order grammars have been recently applied to obtain simpler proofs for the decidability of higher-order (local) model checking [9, 12], and the strictness of tree hierarchy [10]. Haddad [4] developed a type-based transformation to eliminate non-productive OI derivations in deterministic higher-order tree grammars. He has also recently developed a type-based method for logical reflection and selection (which is a kind of grammar transformation) [5]. There is some similarity between the resource λ-calculus [18] and extended terms. In the resource λ-calculus, a function may be applied to a multiset consisting of *linear* terms (which must be used *exactly* once) and *reusable* terms (which may be used *an arbitrary number of times*). In our extended terms, each element of a set must be used *at least* once.

6 Conclusion

We have shown that order-2 unsafe tree languages are context-sensitive, by using novel type-based grammar transformation. It is not yet clear whether this approach can be extended to show context-sensitiveness of languages of arbitrary orders. For the general case, we need to find an appropriate set $\mathcal{C}$ of combinators, and generalize the arguments in Section 4, which are currently specific to the order-2 case. We expect that the grammar transformation in Section 3 is also useful for reasoning about other properties of higher-order languages, such as pumping lemmas for higher-order languages.

Acknowledgments. We thank anonymous reviewers for useful comments. This work was partially supported by JSPS KAKENHI 23220001 and the Mitsubishi Foundation.

References

1. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: Safety is not a restriction at level 2 for string languages. In: Proceedings of FoSSaCS 2005. LNCS, vol. 3441, pp. 490–504. Springer (2005)
2. Curry, H.B., Feys, R.: Combinatory Logic, vol. 1. North-Holland (1958)
3. Damm, W.: The IO- and OI-hierarchies. Theor. Comput. Sci. 20, 95–207 (1982)
4. Haddad, A.: IO vs OI in higher-order recursion schemes. In: Proceedings of FICS 2012. EPTCS, vol. 77, pp. 23–30 (2012)
5. Haddad, A.: Model checking and functional program transformations. In: Proceedings of FSTTCS 2013. LIPIcs, vol. 24, pp. 115–126 (2013)
6. Inaba, K., Maneth, S.: The complexity of tree transducer output languages. In: Proceedings of FSTTCS 2008. LIPIcs, vol. 2, pp. 244–255 (2008)
7. Kartzow, A., Parys, P.: Strictness of the collapsible pushdown hierarchy. In: Proceedings of MFCS 2012. LNCS, vol. 7464, pp. 566–577. Springer (2012)
8. Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Proceedings of FoSSaCS 2002. LNCS, vol. 2303, pp. 205–222. Springer (2002)
9. Kobayashi, N.: Model checking higher-order programs. Journal of the ACM 60(3) (2013)
10. Kobayashi, N.: Pumping by typing. In: Proceedings of LICS 2013. pp. 398–407. IEEE Computer Society (2013)
11. Kobayashi, N., Inaba, K., Tsukada, T.: On unsafe tree and leaf languages. In preparation (2014)
12. Kobayashi, N., Ong, C.-H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: Proceedings of LICS 2009. pp. 179–188. IEEE Computer Society (2009)
13. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Proceedings of PLDI 2011. pp. 222–233 (2011)
14. Kobele, G.M., Salvati, S.: The IO and OI hierarchies revisited. In: Proceedings of ICALP 2013. LNCS, vol. 7966, pp. 336–348. Springer (2013)
15. Maslov, A.N.: The hierarchy of indexed languages of an arbitrary level. Soviet Math. Dokl. 15, 1170–1174 (1974)
16. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: Proceedings of LICS 2006. pp. 81–90. IEEE Computer Society (2006)
17. Ong, C.-H.L., Ramsay, S.: Verifying higher-order programs with pattern-matching algebraic data types. In: Proceedings of POPL 2011. pp. 587–598 (2011)
18. Pagani, M., Rocca, S.R.D.: Solvability in resource lambda-calculus. In: Proceedings of FOSSACS 2010. LNCS, vol. 6014, pp. 358–373. Springer (2010)
19. Turner, R.: An infinite hierarchy of term languages - an approach to mathematical complexity. In: Proceedings of ICALP. pp. 593–608 (1972)
20. Wand, M.: An algebraic formulation of the Chomsky hierarchy. In: Category Theory Applied to Computation and Control. LNCS, vol. 25, pp. 209–213. Springer (1974)