Unsafe Order-2 Tree Languages are Context-Sensitive[1]

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.

[1] An earlier and shorter version of the paper has appeared in Proceedings of FoSSaCS 2014. We have corrected a few errors and restructured the proof since the earlier version.

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.[2]

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 $\{S \to \mathtt{a}AA \mid \mathtt{a} \mid \mathtt{a}A, A \to \mathtt{a}A\mathtt{b} \mid \mathtt{a}\mathtt{b}\}$, 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 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

[2] 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].

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 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.

2 Preliminaries

This section defines higher-order grammars and the languages generated by them. When f is a map, we write $dom(f)$ and $codom(f)$ 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* $A \in dom(\mathcal{N})$ *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:

$$\begin{array}{lll} S \to F\,(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} & F\,h\,x \to h\,x, & F\,h\,x \to F\,(T\,h)\,x \\ C\,h_1\,h_2\,x \to h_1\,x\,x & C\,h_1\,h_2\,x \to h_2\,x\,x & T\,h\,x \to h(h\,x). \end{array}$$

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. We write $e{\downarrow}$ ($f{\downarrow}$, resp.) if e (f, resp.)does not contain any subterms of the form:

1. $\langle \lambda x_1. \cdots \lambda x_k.e \rangle E_1 \cdots E_k$ (where e has type $\mathtt{o}$)
2. $\langle f_1 \rangle (\langle f_2 \rangle E)$.

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 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* $e{\downarrow}$*. (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:

$$\frac{}{[\,] \models a \rightsquigarrow a} \text{ (S-T)} \qquad \frac{}{[\,] \models A \rightsquigarrow A} \text{ (S-NT)}$$

$$\frac{}{[\,] \models \langle f \rangle \rightsquigarrow \langle f \rangle} \text{ (S-C)} \qquad \frac{}{[\{e\}/x] \models x \rightsquigarrow e} \text{ (S-Var)}$$

$$\frac{\theta_1 \models e_1 \rightsquigarrow e_1' \qquad \theta_2 \models E_2 \rightsquigarrow E_2'}{\theta_1 \cup \theta_2 \models e_1\, E_2 \rightsquigarrow e_1'\, E_2'} \text{ (S-App)}$$

$$\frac{\theta_{i,j} \models e_i \rightsquigarrow e_{i,j} \text{ for each } i \in I, j \in J_i}{\bigcup_{i \in I, j \in J_i} \theta_{i,j} \models \{e_i \mid i \in I\} \rightsquigarrow \{e_{i,j} \mid i \in I, j \in J_i\}} \text{ (S-TSet)}$$

Here, the operation $\theta_1 \cup \theta_2$ on substitutions is defined by:

$$dom(\theta_1 \cup \theta_2) = dom(\theta_1) \cup dom(\theta_2)$$
$$(\theta_1 \cup \theta_2)(x) = \begin{cases} \theta_1(x) \cup \theta_2(x) & \text{if } x \in dom(\theta_1) \cap dom(\theta_2) \\ \theta_1(x) & \text{if } x \in dom(\theta_1) \setminus dom(\theta_2) \\ \theta_2(x) & \text{if } x \in dom(\theta_2) \setminus dom(\theta_1) \end{cases}$$

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:

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

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

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

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:

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

$\mathcal{L}(\mathcal{G}_2)$ is the set of all (well-typed)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:

$$\frac{[E_1/x_1, \ldots, E_k/x_k] \models e \rightsquigarrow e'}{\langle \lambda x_1 \cdots x_k.e\rangle\, E_1 \cdots E_k \longrightarrow_\lambda e'} \qquad \text{(RLam-Comb)}$$

$$\frac{[\{\langle f_2\rangle x\}/x] \models f_1 \rightsquigarrow e \qquad e \longrightarrow_\lambda^* e'\downarrow}{\langle \lambda x.\lambda\widetilde{y}.f_1\rangle(\langle f_2\rangle E) \longrightarrow_\lambda \langle \lambda x.\lambda\widetilde{y}.e'\rangle E} \qquad \text{(RLam-Comp)}$$

$$\frac{e \longrightarrow_\lambda e'}{e_0\,(E \cup \{e\}) \longrightarrow_\lambda e_0\,(E \cup \{e'\})} \qquad \text{(RLam-App1)}$$

$$\frac{e_0 \longrightarrow_\lambda e_0'}{e_0\,E \longrightarrow_\lambda e_0'\,E} \qquad \text{(RLam-App2)}$$

In the rules above, the restriction is imposed that every argument of a terminal symbol must be a singleton set; thus, the reduction like $(\lambda x.x\,\{e_1, e_2\})\mathtt{a} \longrightarrow_\lambda \mathtt{a}\,\{e_1, e_2\}$ is not allowed. We write $e \downarrow_\lambda e'$ if $e \longrightarrow_\lambda^* e'\downarrow$.

Henceforth, we assume that every element of $\mathcal{C}$ is fully normalized with respect to $\longrightarrow_\lambda$, i.e., $f\downarrow$ for every $f \in \mathcal{C}$. We also assume that the set $\mathcal{C}$ is closed under composition, in the sense that if $\lambda x.\lambda\widetilde{y}.e_1, f_2 \in \mathcal{C}$ (where e_1 has ground type $\mathtt{o}$) and $[\{\langle f_2\rangle x\}/x] \models f_1 \rightsquigarrow e$ with $e \downarrow_\lambda e'$, then $\lambda x.\lambda\widetilde{y}.e' \in \mathcal{C}$. We write $e \Longrightarrow_{\mathcal{G}} e'$ if $e(\downarrow_\lambda \cdot \longrightarrow_{\mathcal{G}} \cdot\downarrow_\lambda)e'$.

A Variation of Substitutions and Reductions We consider the *linear* version of extended grammars.

$$\begin{array}{l} p \text{ (linear extended terms)} \ ::= a \mid x^{(i)} \mid A^g \mid p\,P \mid \langle g\rangle\,P \\ P ::= (p_1, \ldots, p_k) \qquad g ::= p \mid \lambda(x^{(1)}, \ldots, x^{(k)}) : \gamma_1 \times \cdots \gamma_k.g \\ \gamma \text{ (linear types)} \ ::= \mathtt{o} \mid \gamma \times \cdots \times \gamma \to \gamma \end{array}$$

Here, $k > 0$. We often write $\boldsymbol{x}$ for $(x^{(1)}, \ldots, x^{(k)})$, and $\times\boldsymbol{\gamma}$ for $\gamma_1 \times \cdots \times \gamma_k$. Note that each non-terminal A is now annotated with a (linear extended) λ-term g. It expresses how the non-terminal will be expanded by the rewriting rules. We sometimes omit the annotation when it is not important. As is the case for (non-linear) extended terms, We require that g in $\langle g\rangle$ contains no non-terminals, terminals, nor free variables.

The partial operation $\overline{\cdot}$ defined as follows maps linear types to simple types.

$$\begin{array}{l}\overline{\mathtt{o}} = \mathtt{o} \\ \overline{\gamma_1 \times \cdots \times \gamma_k} = \overline{\gamma_1} \text{ if } \overline{\gamma_1} = \cdots = \overline{\gamma_k} \\ \overline{\times\boldsymbol{\gamma} \to \gamma} = \overline{\times\boldsymbol{\gamma}} \to \overline{\gamma}\end{array}$$

Henceforth we consider only types γ such that $\overline{\gamma}$ is defined. The extended term $\overline{p}$, obtained by collapsing variables and tuples, is defined by:

$$\begin{array}{l}\overline{a} = a \qquad \overline{x^{(i)}} = x \qquad \overline{A^g} = A \qquad \overline{pP} = \overline{p}\overline{P} \\ \overline{\langle g\rangle E} = \langle \overline{g}\rangle \overline{E} \\ \dfrac{\overline{(p_1, \ldots, p_k)} = \{\overline{p_1}, \ldots, \overline{p_k}\}}{\overline{\lambda(x^{(1)}, \ldots, x^{(k)}) : \gamma_1 \times \cdots \times \gamma_k . g} = \lambda x : \overline{\gamma_1 \times \cdots \times \gamma_k} . \overline{g}}\end{array}$$

The type judgment relation $\Delta \vdash p : \gamma$ for linear extended terms is given by:

$$\frac{}{\{x : \gamma\} \vdash x : \gamma} \qquad \frac{}{\emptyset \vdash_E a : \underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{\Sigma(a)} \to \mathtt{o}} \qquad \frac{\emptyset \vdash g : \gamma}{\emptyset \vdash \langle g\rangle : \gamma}$$

$$\frac{A\,\overline{\boldsymbol{x_1}} \cdots \overline{\boldsymbol{x_k}} \to \overline{p} \in \mathcal{R} \qquad \emptyset \vdash \lambda \boldsymbol{x_1} . \cdots \lambda \boldsymbol{x_k} . p : \gamma \qquad \overline{\gamma} = \mathcal{N}(A)}{\emptyset \vdash A^{\lambda \boldsymbol{x_1} . \cdots \lambda \boldsymbol{x_k} . p} : \gamma}$$

$$\frac{\Delta \uplus \{x^{(1)} : \gamma_1, \ldots, x^{(k)} : \gamma_k\} \vdash g : \gamma}{\Delta \vdash \lambda(x^{(1)}, \ldots, x^{(k)}) : \gamma_1 \times \cdots \times \gamma_k . g : \gamma_1 \times \cdots \times \gamma_k \to \gamma}$$

$$\frac{\Delta_i \vdash p_i : \gamma_i \text{ for each } i \in \{1, \ldots, k\}}{\Delta_1 \uplus \cdots \uplus \Delta_k \vdash (p_1, \ldots, p_k) : \gamma_1 \times \cdots \times \gamma_k}$$

$$\frac{\Delta_0 \vdash p : \gamma_1 \times \cdots \times \gamma_k \to \gamma \qquad \Delta_1 \vdash P : \gamma_1 \times \cdots \times \gamma_k}{\Delta_0 \uplus \Delta_1 \vdash pP : \gamma}$$

Here, $\Delta_0 \uplus \Delta_1$ is defined to be $\Delta_0 \cup \Delta_1$ only if $dom(\Delta_0) \cap dom(\Delta_1) = \emptyset$. Note that the rules above require that every variable occurs just once.

We also define the linear version of reductions: $\longrightarrow_{\lambda,\mathtt{lin}}$, $\longrightarrow_{\mathcal{G},\mathtt{lin}}$, and $\Longrightarrow_{\mathcal{G},\mathtt{lin}}$. The relation $\longrightarrow_{\lambda,\mathtt{lin}}$ is defined by:

$$\frac{}{\langle \lambda \boldsymbol{x_1} \cdots \boldsymbol{x_k} . p\rangle\, P_1\ \cdots\ P_k \longrightarrow_{\lambda,\mathtt{lin}} [P_1/\boldsymbol{x_1}, \ldots, P_k/\boldsymbol{x_k}]p} \quad \text{(RLamL-Comb)}$$

Note that, thanks to the linearity, the substitution is now deterministic.

$$\frac{\begin{array}{c}[(\langle f_{2,1}\rangle \boldsymbol{z}_1, \ldots, \langle f_{2,k}\rangle \boldsymbol{z}_k)/\boldsymbol{x}]f_1 \longrightarrow^*_{\lambda,\mathtt{lin}} p' \\ \overline{p'}\downarrow \qquad \overline{(\langle f_{2,1}\rangle P_1, \ldots, \langle f_{2,k}\rangle P_k)} = \{\langle f\rangle E\}\end{array}}{\langle \lambda \boldsymbol{x} . \lambda \widetilde{\boldsymbol{y}} . f_1\rangle(\langle f_{2,1}\rangle P_1, \ldots, \langle f_{2,k}\rangle P_k) \longrightarrow_{\lambda,\mathtt{lin}} \langle \lambda(\boldsymbol{z}_1 + \cdots + \boldsymbol{z}_k) . \lambda \widetilde{\boldsymbol{y}} . p'\rangle(P_1 + \cdots + P_k)} \quad \text{(RLamL-Comp)}$$

Here, we write + for the concatenation operator on tuples.

$$\frac{p_i \longrightarrow_{\lambda,\mathtt{lin}} p_i'}{p_0\,(p_1,\ldots,p_i,\ldots,p_k) \longrightarrow_{\lambda,\mathtt{lin}} p_0\,(p_1,\ldots,p_i',\ldots,p_k)} \quad \text{(RLamL-App1)}$$

$$\frac{p_0 \longrightarrow_{\lambda,\mathtt{lin}} p_0'}{p_0\,P \longrightarrow_{\lambda,\mathtt{lin}} p_0'\,P} \quad \text{(RLamL-App2)}$$

We write $p \downarrow_{\lambda,\mathtt{lin}} p'$ if $p \longrightarrow^*_{\lambda,\mathtt{lin}} p'$ and $\overline{p'}\downarrow$.

The reduction rules are modified accordingly:

$$\frac{}{A^{\lambda \boldsymbol{x_1}.\cdots\lambda \boldsymbol{x_k}.p}\,P_1\,\cdots\,P_k \longrightarrow_{\mathcal{G},\mathtt{lin}} [P_1/\boldsymbol{x_1},\ldots,P_k/\boldsymbol{x_k}]p} \quad \text{(ERL-NT)}$$

$$\frac{p_i \longrightarrow_{\mathcal{G},\mathtt{lin}} p_i' \qquad i \in \{1,\ldots,\Sigma(a)\}}{a\,(p_1)\,\cdots\,(p_{\Sigma(a)}) \longrightarrow_{\mathcal{G},\mathtt{lin}} a\,(p_1)\,\cdots\,(p_i')\,\cdots\,(p_{\Sigma(a)})} \quad \text{(ERL-Cong)}$$

In the rule ERL-NT, $P/\boldsymbol{x}$ abbreviates $p_1/x^{(1)},\ldots,p_k/x^{(k)}$, and the reduction is allowed only when types are preserved. We write $\Longrightarrow_{\mathcal{G},\mathtt{lin}}$ for $\downarrow_{\lambda,\mathtt{lin}}\cdot \longrightarrow_{\mathcal{G},\mathtt{lin}} \cdot\downarrow_{\lambda,\mathtt{lin}}$.

We write $\mathcal{L}_{\Longrightarrow}(\mathcal{G})$ and $\mathcal{L}_{\Longrightarrow_{\mathtt{lin}}}(\mathcal{G})$ for $\{\pi \mid S \Longrightarrow^*_{\mathcal{G}} \pi\}$ and $\{\pi \mid S \Longrightarrow^*_{\mathcal{G},\mathtt{lin}} \pi\}$ respectively. We also write $\mathcal{L}_{\Longrightarrow_{\mathtt{lin},2}}(\mathcal{G})$ for the restriction of $\mathcal{L}_{\Longrightarrow_{\mathtt{lin}}}(\mathcal{G})$ such that for each reduction step $p \Longrightarrow'_{p,\mathtt{lin}}$ (i.e., $p \longrightarrow_{\mathcal{G},\mathtt{lin}} p_0 \longrightarrow_{\lambda,\mathtt{lin}} \mathcal{G}p_1 \longrightarrow_{\lambda,\mathtt{lin}} \mathcal{G}\cdots \longrightarrow_{\lambda,\mathtt{lin}} \mathcal{G}p_n = p'$), it is required that each intermediate term does not contain any *three* consecutive application of combinators (i.e., a subterm of the form $\langle g_1\rangle(\langle g_2\rangle(\langle g_3\rangle p''))$).

The following properties follow immeidately from the definitions.

Lemma 1. *If $e \longrightarrow_{\mathcal{G},\mathtt{lin}} e'$, then $\overline{e} \longrightarrow_{\mathcal{G}} \overline{e'}$. If $e \longrightarrow_{\lambda,\mathtt{lin}} e'$, then $\overline{e} \longrightarrow_{\lambda} \overline{e'}$.*

3.2 From Grammars to Extended Grammars

This section presents a translation from (ordinary) grammars to extended grammars over a set $\mathcal{C}$ of combinators, and shows that the translation preserves the tree language. Here, we assume that for each simple type κ, the set $\mathcal{C}_\kappa = \{f \in \mathcal{C} \mid \emptyset \vdash \langle f\rangle : \kappa\}$ is finite, i.e., there are only finitely many elements of $\mathcal{C}$ that have type κ. 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:*

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

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

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 $\mathtt{o}$ 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 $(\sigma_1 \to \cdots \to \sigma_k \to \mathtt{o}, \eta)$ describes functions that take arguments of types $\sigma_1, \ldots, \sigma_k$ and return a tree of type $\mathtt{o}$. 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 $(\top \to \mathtt{o} \to \mathtt{o}, \langle \lambda y.y \rangle)$, 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.

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

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

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

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

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

Lemma 2. *Supose that $\mathcal{C}_\kappa$ is finite for every κ. Then, the set $\mathbf{ITypes}_\kappa = \{\tau \mid \tau :: \kappa\}$ is finite for every κ.*

Proof. This follows by straightforward induction on the structure of κ. □

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:

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

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

$$\frac{\begin{array}{c} A\, x_1 \cdots x_k \to t \in \mathcal{R} \qquad x_1 : \sigma_1, \ldots, x_k : \sigma_k \vdash t : \mathtt{o} \Rightarrow e \qquad e \downarrow_\lambda e' \\ f = \lambda \mathbf{Vars}(\{x_1 : \sigma_1, \ldots, x_k : \sigma_k\}, x_1 \cdots x_k).e' \in \mathcal{C} \qquad \tau = (\sigma_1 \to \cdots \to \sigma_k \to \mathtt{o}, \langle f \rangle) \end{array}}{\emptyset \vdash A : \tau \Rightarrow \langle f \rangle} \quad \text{(X-NTC)}$$

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

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

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:

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

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

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

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.)

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

The linear version of the production sequence is:

$$\begin{aligned}&S^{g_0} \longrightarrow F^{g_1}\,(\langle g\rangle\,\mathtt{a}, \langle g\rangle\,\mathtt{b}, \langle g\rangle\,\mathtt{a})\,(\mathtt{e},\mathtt{e},\mathtt{e},\mathtt{e}) \longrightarrow T^{g_2}\,(\langle g\rangle\,\mathtt{a}, \langle g\rangle\,\mathtt{b}, \langle g\rangle\,\mathtt{a})\,(\mathtt{e},\mathtt{e},\mathtt{e},\mathtt{e})\\ &\longrightarrow \langle g\rangle\,\mathtt{a}(\langle g\rangle\,\mathtt{b}\,(\mathtt{e},\mathtt{e}), \langle g\rangle\,\mathtt{a}\,(\mathtt{e},\mathtt{e}))\\ &\longrightarrow \mathtt{a}\,(\langle g\rangle\,\mathtt{b}\,(\mathtt{e},\mathtt{e}))\,(\langle g\rangle\,\mathtt{a}\,(\mathtt{e},\mathtt{e})) \longrightarrow^* \mathtt{a}\,(\mathtt{b}\,\mathtt{e}\,\mathtt{e})\,(\mathtt{a}\,\mathtt{e}\,\mathtt{e}).\end{aligned}$$

Here, g, g_0, g_1, g_2 are:

$$\begin{aligned}g &= \lambda h.\lambda(x^{(1)}, x^{(2)}).h\,x^{(1)}\,x^{(2)}\\ g_0 &= F^{g_1}\,(\langle g\rangle\,\mathtt{a}, \langle g\rangle\,\mathtt{b}, \langle g\rangle\,\mathtt{a})\,(\mathtt{e},\mathtt{e},\mathtt{e},\mathtt{e})\\ g_1 &= \lambda(h^{(1)}, h^{(2)}, h^{(3)}).\lambda(x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}).T^{g_2}(h^{(1)}, h^{(2)}, h^{(3)})\,(x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)})\\ g_2 &= \lambda(h^{(1)}, h^{(2)}, h^{(3)}).\lambda(x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}).h^{(1)}(h^{(2)}(x^{(1)}, x^{(2)}), h^{(3)}(x^{(3)}, x^{(4)})).\end{aligned}$$

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 $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{\Longrightarrow}\mathcal{G}' = \mathcal{L}_{\Longrightarrow_{\mathtt{lin}}}(\mathcal{G}') = \mathcal{L}_{\Longrightarrow_{\mathtt{lin},2}}(\mathcal{G}')$.*

Proof. $\mathcal{L}_{\Longrightarrow}\mathcal{G}' \supseteq \mathcal{L}_{\Longrightarrow_{\mathtt{lin}}}(\mathcal{G}')$ follows from Lemma 1 and $\mathcal{L}_{\Longrightarrow_{\mathtt{lin}}}(\mathcal{G}') \supseteq \mathcal{L}_{\Longrightarrow_{\mathtt{lin},2}}(\mathcal{G}')$ follows from the definition. The other inclusions are shown in Appendix B. □

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:

$$\begin{aligned}&\{\lambda x : \mathtt{o}.x\} \cup\\ &\{\lambda y_1 \cdots y_k.y_i\,E_1\,\cdots\,E_\ell \mid E_1 \cup \cdots \cup E_\ell = \{y_1, \ldots, y_k\} \setminus \{y_i\}\}.\end{aligned}$$

Then $\mathcal{C}_\kappa$ is finite for each κ. We shall show that for an extended order-2 grammar over $\mathcal{C}$, if $\pi \in \mathcal{L}(\mathcal{G})$, then there exists a production sequence of π where the size

of intermediate terms 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 for the set-version of extended terms. The size $|(|p)$ of a linear extended term is defined similarly:

$$|a| = |x| = |A| = 1$$
$$|p\,P| = |p| + |P|$$
$$|\langle g \rangle\,P| = 1 + |P|$$
$$|(p_1, \ldots, p_k)| = |p_1| + \cdots + |p_k|$$

Note that in both cases, the size of $\langle f \rangle$ or $\langle g \rangle$ is counted as 1, irrespectively how large f or g is. For extended terms, this is justified by the fact that given a fixed grammar, the number of combinators is fixed. The size of a linear extended term is used just for evaluating that of the corresponding extended term; so it does not need to reflect the memory size required for representing the linear extended term.

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 tree grammar, and $\vdash_{\mathcal{C}} \mathcal{G} \Rightarrow \mathcal{G}'$. If $\pi \in \mathcal{L}(\mathcal{G})$, then there exists an (effectively computable) constant c such that for every tree $\pi \in \mathcal{L}(\mathcal{G})$, there exists a reduction sequence $S = e_0 \Longrightarrow_{\mathcal{G}'} e_1 \Longrightarrow_{\mathcal{G}'} \cdots \Longrightarrow_{\mathcal{G}'} e_n = \pi$ such that for every intermediate term e (including not only e_i but also those occurring in $e_i \Longrightarrow_{\mathcal{G}'} e_{i+1}$), $|e| \leq c|\pi|$ holds.*

As a corollary, the following main result of this paper is obtained.

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

Proof. By Theorem 1, we can effectively construct an order-2 extended grammar $\mathcal{G}'$ over $\mathcal{C}$ such that $\vdash_{\mathcal{C}} \mathcal{G} \Rightarrow \mathcal{G}'$. 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 the numbers of relevant non-terminals, terminals, and combinators are also fixed to be finite values), 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**. By Theorem 1, there exists a reduction sequence $S_{\mathtt{o}} = p_0 \Longrightarrow_{\mathcal{G},\mathtt{lin}} p_1 \Longrightarrow_{\mathcal{G},\mathtt{lin}} \cdots \Longrightarrow_{\mathcal{G},\mathtt{lin}} p_n = \pi$ where no intermediate term contains more than two consecutive applications of combinators; hence, there also exists a corresponding reduction sequence $S_{\mathtt{o}} = \overline{p_0} \Longrightarrow_{\mathcal{G}} \overline{p_1} \Longrightarrow_{\mathcal{G}} \cdots \Longrightarrow_{\mathcal{G}} \overline{p_n} = \pi$ by Lemma 1. Since $|\overline{p}| \leq |p|$, it suffices to show that for every intermediate term p, $|p|$ is linearly bounded by $|\pi|$. It is proved in two steps. In the first step, we show:

Lemma 3. *There exists a constant c_1 such that if $\emptyset \vdash p : \mathtt{o}$ and $p \downarrow_{\lambda,\mathtt{lin}} p'$ and p does not contain more than two consecutive applications of combinators, then $|p| \leq c_1|p'|$*

The above lemma is obtained by a combinatorial argument on the number of combinators that may occur in p. See Appendix C for the proof. Note that the first step does not depend on the choice of $\mathcal{C}$.

In the second step, we show that the size of a linear extended term in normal form (with respect to $\longrightarrow_{\lambda,\mathtt{lin}}$) can be linearly bounded by the size of the tree π generated by the term.

Lemma 4. *There exists a constant c_2 such that if $p\downarrow_{\mathtt{lin}}$ and $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^*\pi$, then $|p| \leq c_2|\pi|$.*

We sketch the proof of the lemma below. See Appendix C for details.

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

Definition 7. *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 $\Delta \vdash p : \gamma \Rightarrow u \dashv \Delta'$ and $\Delta \vdash P : \gamma_1 \times \cdots \times \gamma_k \Rightarrow U \dashv \Delta'$ are defined by the rules below.

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

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

$$\frac{\emptyset \vdash \lambda \boldsymbol{x_1}. \cdots \lambda \boldsymbol{x_k}.p : \gamma \Rightarrow u \dashv \Delta \qquad A\,\overline{x_1}\,\cdots\,\overline{x_k} \to \overline{p} \in \mathcal{R}}{\emptyset \vdash A^{\lambda \boldsymbol{x_1}. \cdots \lambda \boldsymbol{x_k}.p} : \gamma \Rightarrow u \dashv \Delta} \quad \text{(LX-NT)}$$

$$\frac{\Delta_i \vdash p_i : \gamma_i \Rightarrow u_i \dashv \Delta'_i \text{ for each } i \in \{1, \ldots, \ell\}}{\Delta_1 \uplus \cdots \uplus \Delta_\ell \vdash (p_1, \ldots, p_\ell) : \gamma_1 \times \cdots \times \gamma_\ell \Rightarrow (u_1, \ldots, u_\ell) :\dashv \Delta'_1 \uplus \cdots \uplus \Delta'_\ell} \quad \text{(LX-Tup)}$$

$$\frac{\begin{array}{c}\Delta_0 \vdash p_0 : :\gamma_1 \times \cdots \times \gamma_k \to \gamma \Rightarrow u_0 \dashv \Delta'_0 \\ \Delta_1 \vdash P : \gamma_1 \times \cdots \times \gamma_k \Rightarrow U \dashv \Delta'_1\end{array}}{\Delta_0 \uplus \Delta_1 \vdash p_0\, P : \gamma \Rightarrow u_0\, U \dashv \Delta'_0 \uplus \Delta'_1} \quad \text{(LX-App)}$$

$$\frac{\begin{array}{c}\Delta \uplus \{x^{(i_1)} : \gamma_1, \ldots, x^{(i_\ell)} : \gamma_\ell\} \vdash p : \gamma \Rightarrow u \dashv \Delta', x^{(i_1)} : \gamma_1, \ldots, x^{(i_\ell)} : \gamma_\ell \\ x \notin dom(\Delta) \cup dom(\Delta')\end{array}}{\Delta \vdash \lambda \boldsymbol{x}.p : \gamma_1 \times \cdots \times \gamma_\ell \to \gamma \Rightarrow \lambda \boldsymbol{x}.u \dashv \Delta'} \quad \text{(LX-Ab)}$$

If $\Delta \vdash p : \gamma \Rightarrow u \dashv \Delta'$, then u is different from p only in that each non-terminal is replaced by the corresponding λ-term (specified by the annotation for the non-terminal), and that terminal symbols are replaced with variables.

Example 6. Recall $\mathcal{G}'_0$ in Example 5. The term

$$T^{g_2}\,(\langle g \rangle\,\mathsf{a}, \langle g \rangle\,\mathsf{b}, \langle g \rangle\,\mathsf{a})\,(\mathsf{e}, \mathsf{e}, \mathsf{e}, \mathsf{e})$$

occurring in (the linear version of) the production of $\mathsf{a}\,(\mathsf{b}\,\mathsf{e}\,\mathsf{e})\,(\mathsf{a}\,\mathsf{e}\,\mathsf{e})$ is transformed to:

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

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

The transformation satisfies the following property.

Lemma 5. *If $\emptyset \vdash p : \mathtt{o}$ and $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^*\pi$, then there exists u such that $\emptyset \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta$ where for each terminal symbol a, $\Delta \vdash u : \mathtt{o}$ 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 lemma.

Lemma 6. *Let $\mathcal{G}$ be an order-2 extended grammar over $\mathcal{C}$. If $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^*\pi$ and $p{\downarrow_{\mathtt{lin}}}$, 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) $asize(M) \geq |p|$; (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 $p \Longrightarrow^*_{\mathcal{G},\mathtt{lin}} \pi$, one can construct a term u that satisfies the condition of Lemma 5. 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. □

Finally, we show that $asize(M) \leq 28|\{x \mid x : \mathtt{o} \in \Delta\}|$ holds for any pure linear λ-term M and type environment Δ that satisfy the conditions (i), (ii), (iv), and (v). Thus, we have Lemma 4 for $c_2 = 28$. Now we can conclude that if $S_{\mathtt{o}} = p_0 \Longrightarrow_{\mathcal{G},\mathtt{lin}} p_1 \Longrightarrow_{\mathcal{G},\mathtt{lin}} \cdots \Longrightarrow_{\mathcal{G},\mathtt{lin}} p_n = \pi$, then for every intermediate term p in $p_{i-1} \Longrightarrow_{\mathcal{G},\mathtt{lin}} p_i$, $|p|$ is linearly bounded by $|\pi|$, since

$$|p| \leq c_1|p_i| \leq c_1c_2|\pi|.$$

Therefore we have Theorem 2.

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 would like to thank Pawel Parys for spotting errors in an earlier version of the paper. 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)

Appendix

A Examples of Grammar Transformation

We first describe more details about the transformation in Example 5. Recall $\mathcal{G}_0$ in Example 1. Let $\mathcal{C} = \{\lambda g.\lambda x.g\,x, \lambda g.\lambda x.g\,x\,x\}$. Then, non-terminal C can be transformed as follows.

$$\frac{g : (\mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathbf{nc}), x : \mathtt{o} \vdash g_{(\mathtt{o}\to\mathtt{o}\to\mathtt{o},\mathbf{nc})}\, x_{\mathtt{o}}\, x_{\mathtt{o}} : \mathtt{o} \Rightarrow g\,x\,x}{\emptyset \vdash C : ((\mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathbf{nc}) \to \top \to \mathtt{o} \to \mathtt{o}, \langle \lambda g.\lambda x.g\,x\,x\rangle) \Rightarrow \langle \lambda g.\lambda x.g\,x\,x\rangle}$$

$$\frac{g : (\mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathbf{nc}), x : \mathtt{o} \vdash g_{(\mathtt{o}\to\mathtt{o}\to\mathtt{o},\mathbf{nc})}\, x_{\mathtt{o}}\, x_{\mathtt{o}} : \mathtt{o} \Rightarrow g\,x\,x}{\emptyset \vdash C : (\top \to (\mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathbf{nc}) \to \mathtt{o} \to \mathtt{o}, \langle \lambda g.\lambda x.g\,x\,x\rangle) \Rightarrow \langle \lambda g.\lambda x.g\,x\,x\rangle}$$

For F, we have:

$$\frac{g : (\mathtt{o} \to \mathtt{o}, \mathbf{nc}), x : \mathtt{o} \vdash g\,x : \mathtt{o} \Rightarrow g_{(\mathtt{o}\to\mathtt{o},\mathbf{nc})}\, x_{\mathtt{o}}}{\emptyset \vdash F : ((\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \to \mathtt{o} \to \mathtt{o}, \langle \lambda g.\lambda x.g\,x\rangle) \Rightarrow \langle \lambda g.\lambda x.g\,x\rangle}$$

$$\frac{}{\emptyset \vdash F : ((\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \to \mathtt{o} \to \mathtt{o}, \mathbf{nc}) \Rightarrow F_{((\mathtt{o}\to\mathtt{o},\mathbf{nc})\to\mathtt{o}\to\mathtt{o},\mathbf{nc})}}$$

The body $F(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e}$ may be transformed as follows.

$$\dfrac{\dfrac{\dfrac{\emptyset \vdash F(C\,\mathtt{a}\,\mathtt{b}) : (\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \Rightarrow \langle \lambda g.\lambda x.g\,x\rangle\{\langle \lambda g.\lambda x.g\,x\,x\rangle\,\mathtt{a}\} \quad \emptyset \vdash \mathtt{e} : \mathtt{o} \Rightarrow \{\mathtt{e}\}}{\emptyset \vdash F(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} : \mathtt{o} \Rightarrow \langle \lambda g.\lambda x.g\,x\rangle\{\langle \lambda g.\lambda x.g\,x\,x\rangle\,\mathtt{a}\}\,\{\mathtt{e}\}}\ \text{X-RED}}{\emptyset \vdash F(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} : \mathtt{o} \Rightarrow \langle \lambda g.\lambda x.g\,x\,x\rangle\,\mathtt{a}\,\mathtt{e}}\ \text{X-RED}}{\emptyset \vdash F(C\,\mathtt{a}\,\mathtt{b})\,\mathtt{e} : \mathtt{o} \Rightarrow \mathtt{a}\,\mathtt{e}\,\mathtt{e}}$$

Here, $\emptyset \vdash F(C\,\mathtt{a}\,\mathtt{b}) : (\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \Rightarrow \langle \lambda g.\lambda x.g\,x\rangle\{\langle \lambda g.\lambda x.g\,x\,x\rangle\,\mathtt{a}\}$ can be obtained from the transformation for F above and

$$\emptyset \vdash C\,\mathtt{a}\,\mathtt{b} : (\mathtt{o} \to \mathtt{o}, \mathbf{nc}) \Rightarrow \{\langle \lambda g.\lambda x.g\,x\,x\rangle\,\mathtt{a}\}.$$

From the above transformations, we obtain the rule $S_\mathtt{o} \to \mathtt{a}\,\mathtt{e}\,\mathtt{e}$. The whole grammar $\mathcal{G}'_0$ is as given in Example 5.

Next, recall Example 2. Here are some of the rules obtained by the transformation. All the flags in the types below are **nc**, hence omitted. (The whole rules are too many to be listed here.)

$S_\mathtt{o} \to F_{\top\to\top\to\top\to\mathtt{o}}$
$S_\mathtt{o} \to F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ G_{\mathtt{o}\to\mathtt{o}}\ \mathtt{a}\ \mathtt{b}$
$G_{\mathtt{o}\to\mathtt{o}}\,x \to \mathtt{g}\ x\ \mathtt{e}$
$H_{\mathtt{o}\to\mathtt{o}}\,x \to \mathtt{g}\ \mathtt{e}\ x$
$F_{\top\to\top\to\top\to\mathtt{o}} \to \mathtt{e}$
$F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ F_{\top\to\top\to\top\to\mathtt{o}}\ (\mathtt{f}\ (\varphi\ y)\ x)$
$F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ \varphi\ x \to \mathtt{f}\ F_{\top\to\top\to\top\to\mathtt{o}}\ (\mathtt{f}\ \varphi\ x)$
$F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ (F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ (F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x)\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ (\varphi\ y)\ x)$
$F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ (F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x)\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ \varphi\ x)$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\ y \to \mathtt{f}\ (F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ E\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ \varphi_{\top\to\mathtt{o}}\ \ x)$
for each $E \subseteq \{F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ x, F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x,$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\}$ that contains $\varphi_{\mathtt{o}\to\mathtt{o}}$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\ y \to \mathtt{f}\ (F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ E\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ (\varphi_{\mathtt{o}\to\mathtt{o}}\ y)\ \ x)$
for each $E \subseteq \{F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ x, F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x,$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\}$ that contains $\varphi_{\top\to\mathtt{o}}$
$F_{(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ (F_{\top\to\top\to\top\to\mathtt{o}})\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ (\varphi\ y)\ x)$
$F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ E\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ \varphi\ x)$
for each non-empty $E \subseteq \{F_{\top\to\top\to\top\to\mathtt{o}}, F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ \varphi\ x\}$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\ y \to \mathtt{f}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ E\ y\ (H_{\mathtt{o}\to\mathtt{o}}\ y))\ (\mathtt{f}\ (\varphi_{\mathtt{o}\to\mathtt{o}}\ y)\ x)$
for each $E \subseteq \{F_{\top\to\top\to\top\to\mathtt{o}}, F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ x\}$ that contains $\varphi_{\top\to\mathtt{o}}$
$F_{(\top\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi\ x\ y \to \mathtt{f}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ E\ y)\ (\mathtt{f}\ \varphi\ x)$
for each non-empty $E \subseteq \{F_{\top\to\top\to\top\to\mathtt{o}}, F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ \varphi\ x\}$
$F_{(\top\to\mathtt{o})\wedge(\mathtt{o}\to\mathtt{o})\to\mathtt{o}\to\mathtt{o}\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ \varphi_{\mathtt{o}\to\mathtt{o}}\ x\ y \to \mathtt{f}\ (F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ E\ y)\ (\mathtt{f}\ (\varphi_{\mathtt{o}\to\mathtt{o}}\ y)\ x)$
for each $E \subseteq \{F_{\top\to\top\to\top\to\mathtt{o}}, F_{(\top\to\mathtt{o})\to\mathtt{o}\to\top\to\mathtt{o}}\ \varphi_{\top\to\mathtt{o}}\ x\}$ that contains $\varphi_{\top\to\mathtt{o}}$
...

B Proof of Theorem 1

Theorem 1 follows from the following three theorems, which are proved in the following subsections.

Theorem 3. *Suppose that* $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$ *for* $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$. *Then* $\mathcal{G}'$ *is a valid extended grammar.*

Theorem 4. *Suppose that* $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$ *for* $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ *and* $\mathcal{G}' = (\Sigma', \mathcal{N}', \mathcal{R}', S_\mathtt{o})$. *If* $S_\mathtt{o} \Longrightarrow^*_{\mathcal{G}'} \pi$ *for a tree* π, *then* $S \longrightarrow^*_{\mathcal{G}} \pi$.

Theorem 5. *Suppose that* $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$ *for* $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ *and* $\mathcal{G}' = (\Sigma', \mathcal{N}', \mathcal{R}', S_{\mathtt{o}})$. *If* $S \longrightarrow_{\mathcal{G}}^{*} \pi$ *for a tree* π, *then* $S_{\mathtt{o}} \Longrightarrow_{\mathcal{G}'}^{*} \pi$ *and* $S_{\mathtt{o}} \Longrightarrow_{\mathcal{G}',\mathtt{lin}}^{*} \pi$.

B.1 Proof of Theorem 3

We define $[\![\Gamma]\!]$ by:

$$[\![\emptyset]\!] = \emptyset \qquad [\![\,\Gamma, x:\tau]\!] = \begin{cases} [\![\Gamma]\!], x_\tau : [\![\tau]\!] & \text{if } \mathit{flag}(\tau) = \mathbf{nc} \\ [\![\Gamma]\!] & \text{if } \mathit{flag}(\tau) \neq \mathbf{nc} \end{cases}$$

Lemma 7. *If* $\Gamma \vdash t : \tau \Rightarrow e$, *then* $[\![\Gamma]\!] \vdash e : [\![\tau]\!]$.

Proof. This follows by straightforward induction on the derivation of $\Gamma \vdash t : \tau \Rightarrow e$. □

Lemma 8. *For each triple* (Γ, t, τ), *the number of* e *such that* $\Gamma \vdash t : \tau \Rightarrow e$ *is finite.*

Proof. This follows by straightforward induction on the structure of t. □

Proof of Theorem 3. Suppose that $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$. By the assumption that $\mathcal{C}_\kappa$ is finite for every κ and Lemmas 2 and 8, the set of rules of $\mathcal{G}'$ is finite. Lemma 7 ensures that each rule obtained by the transformation satisfies the requirement on typing (i.e., the condition "$\mathcal{N}(A)$ must be of the form $\kappa_1 \to \cdots \to \kappa_k \to \mathtt{o}$ and $\Gamma \cup \{x_1{:}\kappa_1, \ldots, x_k{:}\kappa_k\} \vdash_E t : \mathtt{o}$ must hold for some $\Gamma \subseteq \mathcal{N}$"). The additional requirements $\lambda x_1. \cdots \lambda x_k.e \notin \mathcal{C}$ and $e{\downarrow}$ are also satisfied by the conditions on the set of rules: $\lambda\mathbf{Vars}(\mathcal{K}, x_1 \cdots x_k).e' \notin \mathcal{C} \wedge e \downarrow_\lambda e'$. □

B.2 Proof of Theorem 4

Suppose $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$. We write $\longrightarrow_{\lambda,\mathtt{w}}$ and $\longrightarrow_{\mathcal{G}',\mathtt{w}}$ for the weaker versions of $\longrightarrow_\lambda$ and $\longrightarrow_{\mathcal{G}'}$, obtained by weakening the substitution relation by adding the rule:

$$\frac{\theta \models e \rightsquigarrow e'}{\theta \cup \theta' \models e \rightsquigarrow e'}$$

We write $e \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} e'$ if it is obtained by applying RLam-Comb to a top-level redex (which is not guarded by non-terminals or variables). Otherwise we write $e \longrightarrow_{\lambda,\mathtt{w},\mathtt{i}} e'$. We write $e \preceq e'$ if $e = e'$ or e is obtained by removing some elements from term sets of e'.

Lemma 9. *There is no infinite outermost reduction sequence* $e \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} e_1 \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} e_2 \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} \cdots$.

Proof. This follows by a standard argument using logical relations. □

Lemma 10. *If* $e \longrightarrow_{\lambda,\mathtt{w},\mathtt{i}} \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} e'$, *then there exists* e'' *such that* $e' \preceq e''$.

Proof. We discuss only the case where

$$e = C[\langle \lambda x.\lambda \widetilde{y}.e_0\rangle(\langle f\rangle E_0)\widetilde{E}] \longrightarrow_{\lambda,\mathtt{w},\mathtt{i}} C[\langle \lambda x.\lambda \widetilde{y}.e_0'\rangle E_0\, \widetilde{E}] \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} C[e']$$

with

$$[\langle f\rangle x/x]e_0 \downarrow_{\lambda,\mathtt{w}} e_0' \qquad [E_0/x, \widetilde{E}/\widetilde{y}] \models e_0' \rightsquigarrow e'.$$

The other cases are obvious, since the redexes do not overlap with each other.

By the condition $[\langle f\rangle x/x]e_0 \downarrow_{\lambda,\mathtt{w}} e_0'$, we have $[\langle f\rangle E_0/x]e_0 \longrightarrow^*_{\lambda,\mathtt{w}} [E_0/x]e_0'$. By this and the condition $[E_0/x, \widetilde{E}/\widetilde{y}] \models e_0' \rightsquigarrow e'$, there exist e'' and e''' such that

$$\begin{array}{ll} [\widetilde{E}/\widetilde{y}] \models e_0 \rightsquigarrow e''' & [\widetilde{E}/\widetilde{y}] \models e_0' \rightsquigarrow e'' \\ e \longrightarrow_{\lambda,\mathtt{w},\mathtt{o}} e''' \longrightarrow^*_{\lambda,\mathtt{w}} e'' & \\ e' \preceq e'', & \end{array}$$

as required. □

Lemma 11. *If $e_1 \longrightarrow_{\mathcal{G}',\mathtt{w}} e_2$ and $e_1 \preceq e_1'$, then $e_1' \longrightarrow_{\mathcal{G}',\mathtt{w}} e_2'$ and $e_2 \preceq e_2'$ for some e_2'. Similarly for $\longrightarrow_{\lambda,\mathtt{w}}$.*

Proof. This follows by straightforward induction on the derivation of $e_1 \longrightarrow_{\mathcal{G}',\mathtt{w}} e_2$ and $e_1 \longrightarrow_{\lambda,\mathtt{w}} e_2$. □

Lemma 12. *If $e(\longrightarrow_{\lambda,\mathtt{w}} \cup \longrightarrow_{\mathcal{G}',\mathtt{w}})^*\pi$, then $e \longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$. Furtheremore, $e \longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$ reduces the same number of non-terminals as $e(\longrightarrow_{\lambda,\mathtt{w}} \cup \longrightarrow_{\mathcal{G}',\mathtt{w}})^*\pi$.*

Proof. If $e \longrightarrow^*_{\lambda,\mathtt{w},\mathtt{i}} e'' \longrightarrow_{\mathcal{G}'} e'$, then by Lemmas 9 and 10, we have $e \longrightarrow^*_{\lambda,\mathtt{w},\mathtt{o}} \longrightarrow_{\mathcal{G}'} \longrightarrow^*_{\lambda,\mathtt{w}} e'''$ with $e' \preceq e'''$ (because the repeated applications of 10 eventually moves the redex for $e'' \longrightarrow_{\mathcal{G}'} e'$ to a top-level position). By repeatedly applying this property and Lemma 11, we get $e \longrightarrow^*_{\mathcal{G}',\mathtt{w}} e'$ with $\pi \subseteq e'$. But then it must be the case that $e' = \pi$. □

The following is an immeidate corollary of the above lemma.

Lemma 13. *If $S \Longrightarrow^*_{\mathcal{G}'} \pi$, then $S \longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$.*

Proof. Suppose $S \Longrightarrow^*_{\mathcal{G}'} \pi$. Then $S \Longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$ follows immediately. By Lemma 12, we have $S \longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$. □

It remains to show that $S \longrightarrow^*_{\mathcal{G}',\mathtt{w}} \pi$ implies $S \longrightarrow^*_{\mathcal{G}} \pi$. We write $e \longrightarrow^{\leq n}_{\mathcal{G}',\mathtt{w}} e'$ if there is a reduction sequence $e \longrightarrow_{\mathcal{G}',\mathtt{w}} \cdots \longrightarrow_{\mathcal{G}',\mathtt{w}} e'$ that uses the rule ER-NT at most n times. We define the relations $\models_n t : \tau \Rightarrow e$, $\models_n t : \tau \Rightarrow E$, and $\models_n t : (\tau_1, \ldots, \tau_\ell) \Rightarrow (E_1, \ldots, E_k)$ (where t is a closed term, e is a closed extended term, and E_i's are sets of closed extended terms) by induction on the structure of τ as follows.

- $\models_n t : \mathtt{o} \Rightarrow e$ iff $e \longrightarrow^{\leq n}_{\mathcal{G}',\mathtt{w}} \pi$ implies $t \longrightarrow^*_{\mathcal{G}} \pi$

- $\models_n\ t : (\bigwedge\{\tau_1,\ldots,\tau_\ell\} \to \rho,\eta) \Rightarrow e$ iff for every $n' \le n$ and $E_1,\ldots,E_k$, $\models_{n'} s : (\tau_1,\ldots,\tau_\ell) \Rightarrow (E_1,\ldots,E_k)$ implies $\models_{n'} ts : (\rho,\eta') \Rightarrow e\,E_1\cdots E_k$, where $\eta' = \eta$ if $\mathit{flag}(\tau_i) \neq \mathbf{nc}$ for every $i \in \{1,\ldots,\ell\}$ and $\eta' = \mathbf{nc}$ otherwise.
- $\models_n t : \tau \Rightarrow \{e_1,\ldots,e_k\}$ iff $\models_n t : \tau \Rightarrow e_i$ for every $i \in \{1,\ldots,k\}$.
- $\models_n t : (\tau_1,\ldots,\tau_\ell) \Rightarrow (E_1,\ldots,E_k)$ iff $\models_n t : \tau_i \Rightarrow E_i$ and $\mathit{flag}(\tau_i) = \mathbf{nc}$ for $i \in \{1,\ldots,k\}$ and $\models_n t : \tau_i \Rightarrow \mathit{flag}(\tau_i)$ and $\mathit{flag}(\tau_i) \neq \mathbf{nc}$ for $i \in \{k{+}1,\ldots,\ell\}$.

We also write $\models_n s : \sigma \Rightarrow (E_1,\ldots,E_k)$ for $\models_n s : (\tau_1,\ldots,\tau_\ell) \Rightarrow (E_1,\ldots,E_k)$, when $\sigma = \bigwedge\{\tau_1,\ldots,\tau_\ell\}$ and $\tau_i < \tau_j$ for every $i < j$ (assuming that the total order $<$ has been chosen so that $\tau < \tau'$ whenever $\mathit{flag}(\tau) = \mathbf{nc}$ and $\mathit{flag}(\tau') \neq \mathbf{nc}$). We write $(\theta_1,\theta_2) \models_n \Gamma$ if (i) $\mathit{dom}(\theta_1) = \mathit{dom}(\Gamma)$, (ii) $\mathit{dom}(\theta_2) = \{x_\tau \mid x : \tau \in \Gamma, \mathit{flag}(\tau) = \mathbf{nc}\}$ (iii) for every $x : \tau \in \Gamma$ with $\mathit{flag}(\tau) = \mathbf{nc}$, $\models_n \theta_1(x) : \tau \Rightarrow \theta_2(x_\tau)$, and (iv) for every $x : \tau \in \Gamma$ with $\mathit{flag}(\tau) \neq \mathbf{nc}$, $\models_n \theta_1(x) : \tau \Rightarrow \mathit{flag}(\tau)$, For an open term t, we write $\Gamma \models_n t : \tau \Rightarrow e$ if $\models_{n'} \theta_1 t : \tau \Rightarrow e'$ holds whenever $(\theta_1,\theta_2) \models_{n'} \Gamma$ and $\theta_2 \models e \rightsquigarrow e'$ with $n' \le n$.

Lemma 14. *If $\Gamma \vdash t : \tau \Rightarrow e$, then $\Gamma \models_n t : \tau \Rightarrow e$ holds for every n. If $\Gamma \vdash t : \tau \Rightarrow E$, then $\Gamma \models_n t : \tau \Rightarrow E$ holds for every n.*

Proof. The proof proceeds by double induction on n and the derivation of $\Gamma \vdash t : \tau \Rightarrow e$ or $\Gamma \vdash t : \tau \Rightarrow E$, with case analysis on the last rule used in the derivation.

- Case X-VarC: In this case, $\Gamma = x : \tau$, $t = x$, and $e = \langle f \rangle$ with $\mathit{flag}(\tau) = \langle f \rangle$. Suppose $(\theta_1,\theta_2) \models_{n'} \Gamma$ and $\theta_2 \models e \rightsquigarrow e'$ with $n' \le n$. Then, it must be the case that $\theta_2 = \emptyset$ and $e' = e = \langle f \rangle$, so we have $\models_{n'} \theta_1(x) : \tau \Rightarrow \langle f \rangle$. Therefore, we have $\Gamma \models_n t : \tau \Rightarrow e$ as required.
- Case X-Var: In this case, $\Gamma = x : \tau$, $t = x$, and $e = x_\tau$ with $\mathit{flag}(\tau) = \mathbf{nc}$. Suppose $(\theta_1,\theta_2) \models_{n'} \Gamma$ and $\theta_2 \models e \rightsquigarrow e'$ with $n' \le n$. Then, it must be the case that $\theta_2 = [\{e'\}/x_\tau]$, and $\models_{n'} \theta_1(x) : \tau \Rightarrow \theta_2(x_\tau)$. Thus, we have $\Gamma \models_n t : \tau \Rightarrow e$ as required.
- Case X-NT: In this case, $\Gamma = \emptyset$, $t = A$, and $u = A_\tau$ with $\mathit{flag}(\tau) = \mathbf{nc}$. It suffices to show $\models_n A : \tau \Rightarrow A_\tau$. Let $\tau = (\bigwedge_{j\in\{1,\ldots,\ell_1\}} \tau_{1,j} \to \cdots \to \bigwedge_{j\in\{1,\ldots,\ell_m\}} \tau_{m,j} \to \mathtt{o}, \mathbf{nc})$ and $\models_{n_i} t_i : (\tau_{i,1},\ldots,\tau_{i,\ell_i}) \Rightarrow (E_{i,1},\ldots,E_{i,k_i})$ with $n \ge n_1 \ge \cdots \ge n_m$, and suppose $A_\tau\, E_{1,1}\cdots E_{1,k_1}\cdots E_{m,1}\cdots E_{m,k_m} \longrightarrow^{\le n_m}_{\mathcal{G}',\mathtt{w}} \pi$. We need to show $A\,t_1\cdots t_m \longrightarrow^*_{\mathcal{G}} \pi$. By the assumption

$$A_\tau\, E_{1,1}\cdots E_{1,k_1}\cdots E_{m,1}\cdots E_{m,k_m} \longrightarrow^{\le n_m}_{\mathcal{G}',\mathtt{w}} \pi,$$

we have:

$$\begin{array}{l} A\,x_1,\ldots,x_m \to t_0 \in \mathcal{R}_{\mathcal{G}} \\ x_1 : \tau_{1,1},\ldots,x_1 : \tau_{1,k_1},\ldots,x_m : \tau_{m,1},\ldots,x_m : \tau_{m,k_m} \vdash t_0 : \mathtt{o} \Rightarrow e_0 \\ [E_{1,k_1}/x_{1,\tau_{1,k_1}},\ldots,E_{m,k_m}/x_{m,\tau_{m,k_m}}] \models e_0 \rightsquigarrow e_0' \\ e_0' \longrightarrow^{\le n_m - 1}_{\mathcal{G}',\mathtt{w}} \pi \end{array}$$

By the induction hypothesis, we have:

$$x_1 : \tau_{1,1},\ldots,x_1 : \tau_{1,k_1},\ldots,x_m : \tau_{m,1},\ldots,x_m : \tau_{m,k_m} \models_{n-1} t_0 : \mathtt{o} \Rightarrow e_0.$$

By the condition $\models_{n_i} t_i : (\tau_{i,1}, \ldots, \tau_{i,\ell_i}) \Rightarrow (E_{i,1}, \ldots, E_{i,k_i})$, we have $(\theta_1, \theta_2) \models_{n_m} x_1{:}\tau_{1,1}, \ldots, x_1{:}\tau_{1,k_1}, \ldots, x_m{:}\tau_{m,1}, \ldots, x_m{:}\tau_{m,k_m}$ for $\theta_1 = [t_1/x_1, \ldots, t_m/x_m]$ and $\theta_2 = [E_{1,k_1}/x_{1,\tau_{1,k_1}}, \ldots, E_{m,k_m}/x_{m,\tau_{m,k_m}}]$. Thus, we have

$$\models_{n_m - 1} [t_1/x_1, \ldots, t_m/x_m]t_0 : \mathtt{o} \Rightarrow e_0'.$$

Since $e_0' \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_m - 1} \pi$, we have $t \longrightarrow_{\mathcal{G}} [t_1/x_1, \ldots, t_m/x_m]t_0 \longrightarrow_{\mathcal{G}}^{*} \pi$ as required.

– Case X-NTC; In this case, we have:

$$\begin{array}{l} \Gamma = \emptyset \qquad t = A \qquad A\, x_1 \cdots x_m \to t_0 \in \mathcal{R} \\ e = \langle \lambda\mathbf{Vars}(x_1 : \sigma_1, \ldots, x_m : \sigma_m, x_1 \cdots x_m).e_0 \rangle \\ \tau = (\sigma_1 \to \cdots \to \sigma_m \to \mathtt{o}, \langle \lambda\mathbf{Vars}(x_1 : \sigma_1, \ldots, x_m : \sigma_m, x_1 \cdots x_m).e_0 \rangle) \\ x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash t_0 : \mathtt{o} \Rightarrow e_0' \\ e_0' \downarrow_\lambda e_0 \end{array}$$

Let $\sigma_i = \bigwedge_{j \in \{1,\ldots,\ell_1\}} \tau_{i,j}$, $\models_{n_i} t_i : (\tau_{i,1}, \ldots, \tau_{i,\ell_i}) \Rightarrow (E_{i,1}, \ldots, E_{i,k_i})$, and suppose $\langle \lambda\mathbf{Vars}(x_1 : \sigma_1, \ldots, x_m : \sigma_m, x_1 \cdots x_m).e_0 \rangle\, E_{1,1} \cdots E_{1,k_1} \cdots E_{m,1} \cdots E_{m,k_m} \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_m} \pi$ where $n \geq n_1 \geq \cdots \geq n_m$. It suffices to show that $A\, t_1 \cdots t_m \longrightarrow_{\mathcal{G}}^{*} \pi$. By the induction hypothesis, we have:

$$x_1 : \sigma_1, \ldots, x_m : \sigma_m \models_n t_0 : \mathtt{o} \Rightarrow e_0'.$$

By Lemma 12 with the conditions

$$\langle \lambda\mathbf{Vars}(x_1 : \sigma_1, \ldots, x_m : \sigma_m, x_1 \cdots x_m).e_0 \rangle\, E_{1,1} \cdots E_{1,k_1} \cdots E_{m,1} \cdots E_{m,k_m} \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_m} \pi$$

and $e_0' \downarrow_\lambda e_0$, we have

$$\langle \lambda\mathbf{Vars}(x_1 : \sigma_1, \ldots, x_m : \sigma_m, x_1 \cdots x_m).e_0' \rangle\, E_{1,1} \cdots E_{1,k_1} \cdots E_{m,1} \cdots E_{m,k_m} \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_m} \pi.$$

Thus, we have e' such that $e' \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_m} \pi$ and $\theta_2 \models e_0' \rightsquigarrow e'$ for $\theta_2 = [E_{1,k_1}/x_{1,\tau_{1,k_1}}, \ldots, E_{m,k_m}/x_{m,\tau_{m,k_m}}]$. Since $(\theta_1, \theta_2) \models_n x_1 : \sigma_1, \ldots, x_m : \sigma_m$ for $\theta_1 = [t_1/x_1, \ldots, t_m/x_m]$, we have $A\, t_1 \cdots t_m \longrightarrow_{\mathcal{G}} \theta_1 t_0 \longrightarrow_{\mathcal{G}}^{*} \pi$ as required.

– Case X-T: In this case, $t = a$ and $e = a$, with $\tau = (\mathtt{o} \to \cdots \to \mathtt{o} \to \mathtt{o}, \mathbf{nc})$. Suppose that $\models_{n_i} t_i : \mathtt{o} \Rightarrow e_i$ and $a\, e_1 \cdots e_{\Sigma(a)} \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n_{\Sigma(a)}} \pi$ with $n \geq n_1 \geq \cdots \geq n_{\Sigma(a)}$. It suffices to show $a\, t_1 \cdots t_{\Sigma(a)} \longrightarrow_{\mathcal{G}}^{*} \pi$. By the condition $a\, e_1 \cdots e_{\Sigma(a)} \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n} \pi$, $\pi = a\, \pi_1 \cdots \pi_{\Sigma(a)}$ with $e_i \longrightarrow_{\overline{\mathcal{G}}',\mathtt{w}}^{\leq n} \pi_i$ for each i. By the assumption $\models_{n_i} t_i : \mathtt{o} \Rightarrow e_i$, we have $t_i \longrightarrow_{\mathcal{G}}^{*} \pi_i$. Thus, we have $t \longrightarrow_{\mathcal{G}}^{*} \pi$ as required.

– Case X-App: In this case, we have:

$$\begin{array}{l} t = t_0 t_1 \qquad e = e_0 E_1 \cdots E_k \qquad \tau = (\rho, \eta') \\ \Gamma = \Gamma_0 \cup (\bigcup_{i \in \{1,\ldots,\ell\}} \Gamma_i) \qquad \Gamma_0 \vdash t_0 : (\bigwedge\{\tau_1, \ldots, \tau_\ell\} \to \rho, \eta) \Rightarrow e_0 \\ \Gamma_i \vdash t_1 : \tau_i \Rightarrow E_i \qquad \mathit{flag}(\tau_i) = \mathbf{nc} \text{ for each } i \in \{1, \ldots, k\} \\ \Gamma_i \vdash t_1 : \tau_i \Rightarrow e_{1,i}' \qquad \mathit{flag}(\tau_i) \neq \mathbf{nc} \text{ (for each } i \in \{k+1, \ldots, \ell\}) \end{array}$$

Suppose that $(\theta_1, \theta_2) \models_n \Gamma$ and $\theta_2 \models e \rightsquigarrow e'$ hold. We need to show $\models_n \theta_1 t : \tau \Rightarrow e'$. By the assumptions, we have $e'_0, E'_1, \ldots, E'_k$, $\theta_{1,0}, \theta_{1,1}, \theta_{2,1}, \theta_{2,2}$ that satisfy the following conditions.

$$\theta_{1,0} \cup \theta_{1,1} = \theta_1 \qquad \theta_2 = \theta_{2,1} \cup \theta_{2,2} \qquad (\theta_{1,0}, \theta_{2,0}) \models_n \Gamma_0 \qquad (\theta_{1,1}, \theta_{2,1}) \models_n \Gamma_1 \cup \cdots \cup \Gamma_\ell$$
$$e' = e'_0 E'_1 \cdots E'_k \qquad \theta_{2,1} \models (E_1, \ldots, E_k) \rightsquigarrow (E'_1, \ldots, E'_k) \qquad \theta_{2,0} \models e_0 \rightsquigarrow e'_0$$

Here, the notation $\theta \models (E_1, \ldots, E_k) \rightsquigarrow (E'_1, \ldots, E'_k)$ means that there are $\theta_1, \ldots, \theta_k$ such that $\theta = \theta_1 \cup \cdots \theta_k$ and $\theta_i \models E_i \rightsquigarrow E'_i$ for each $i \in \{1, \ldots, k\}$. By the induction hypothesis, we have:

$$\Gamma_0 \models_n t_0 : (\bigwedge\{\tau_1, \ldots, \tau_\ell\} \to \rho, \eta) \Rightarrow e_0$$
$$\Gamma_1 \cup \cdots \cup \Gamma_\ell \models_n t_1 : (\tau_1, \ldots, \tau_\ell) \Rightarrow (E_1, \ldots, E_k)$$

Thus, we have

$$\models_n \theta_{1,0} t_0 : (\bigwedge\{\tau_1, \ldots, \tau_\ell\} \to \rho, \eta) \Rightarrow e'_0$$
$$\models_n \theta_{1,1} t_1 : (\tau_1, \ldots, \tau_\ell) \Rightarrow (E'_1, \ldots, E'_k)$$

Thus, we have $\models_n \theta_1 t : \tau \Rightarrow e'$ as required.

– Case X-Set: Trivial by the induction hypothesis.

□

Proof of Theorem 4. This follows immediately from Lemma 13, $\emptyset \vdash S : \mathsf{o} \Rightarrow S_{\mathsf{o}}$ and Lemma 14.
□

B.3 Proof of Theorem 5

Lemma 15. *If $\Gamma \vdash t : \tau \Rightarrow e$ and $\mathit{flag}(\tau) \neq \mathbf{nc}$, then $e = \mathit{flag}(\tau)$ and $[\![\Gamma]\!] = \emptyset$.*

Proof. This follows by straightforward induction on derivation of $\Gamma \vdash t : \tau \Rightarrow e$.
□

We write $\Gamma \vdash_{\mathtt{lin}} t : \tau \Rightarrow p : \gamma \dashv \Delta$ if $\Gamma \vdash t : \tau \Rightarrow \overline{p}$ and $\Delta \vdash p : \gamma$. Similarly, we write $\Gamma \vdash_{\mathtt{lin}} t : \tau \Rightarrow P : \times\boldsymbol{\gamma} \dashv \Delta$ if $\Gamma \vdash t : \tau \Rightarrow \overline{P}$ and $\Delta \vdash P : \times\boldsymbol{\gamma}$. We just write $\Gamma \vdash t : \tau \Rightarrow p$ if $\Gamma \vdash_{\mathtt{lin}} t : \tau \Rightarrow p : \gamma \dashv \Delta$ for some Δ and γ.

Lemma 16 (de-substitution). *If $\Gamma \vdash_{\mathtt{lin}} [t/x]s : \tau \Rightarrow p : \gamma \dashv \Delta$, then*

$$\Gamma_0 \cup \{x : \tau_i \mid i \in I \cup J\} \vdash_{\mathtt{lin}} s : \tau \Rightarrow p_0 : \gamma \dashv \Delta_0 \uplus \{\boldsymbol{x}_{\boldsymbol{\tau_i}} : \boldsymbol{\gamma_i} \mid i \in I\}$$
$$\mathit{flag}(\tau_i) = \mathbf{nc} \text{ for } i \in I \qquad \mathit{flag}(\tau_i) \neq \mathbf{nc} \text{ for } i \in J$$
$$\Gamma_i \vdash_{\mathtt{lin}} t : \tau_i \Rightarrow P_i : \times\boldsymbol{\gamma_i} \dashv \Delta_i \text{ for } i \in I \qquad \Gamma_i \vdash_{\mathtt{lin}} t : \tau_i \Rightarrow p_i \text{ for } i \in J$$
$$p = [P_i/\boldsymbol{x}_{\boldsymbol{\tau_i}}]_{i \in I} p_0$$
$$\Gamma \supseteq \bigcup_{i \in \{0\} \cup I \cup J} \Gamma_i \qquad \Delta = \biguplus_{i \in \{0\} \cup I} \Delta_i$$

hold for some I, J, Γ_i(for $i \in \{0\} \cup I \cup J$), τ_i(for $i \in I \cup J$), $\boldsymbol{x}_{\boldsymbol{\tau_i}}, P_i, \boldsymbol{\gamma_i}$(for $i \in I$), and Δ_i, p_i(for $i \in \{0\} \cup J$).

Proof. The proof proceeds by induction on the structure of s.

- Case $s = x$: If $\mathit{flag}(\tau) = \mathbf{nc}$, then the result holds for:

$$\begin{array}{llllll} p_0 = x_\tau^{(1)} & I = \{1\} & J = \emptyset & \tau_1 = \tau & P_1 = (p) \\ \Gamma_1 = \Gamma & \Gamma_0 = \Delta_0 = \emptyset & \boldsymbol{\gamma_1} = (\gamma) & \Delta_1 = \Delta \end{array}$$

  If $\mathit{flag}(\tau) \neq \mathbf{nc}$, then by Lemma 15, $\overline{p} = \mathit{flag}(\tau)$ and $\Delta = \emptyset$. Thus, the result holds for:

$$p_0 = p_1 = p \qquad I = \emptyset \qquad J = \{1\} \qquad \tau_1' = \tau \qquad \Gamma_1 = \Gamma \qquad \Gamma_0 = \Delta_0 = \emptyset$$

- Case $s = y \neq x$, $s = a$ or $s = A$: In this case, we have $\Gamma \vdash_{\mathtt{lin}} s : \tau \Rightarrow p$. The result holds for:

$$I = J = \emptyset \qquad p_0 = p \qquad \Gamma_0 = \Gamma \qquad \Delta_0 = \Delta$$

- Case $s = s_1 s_2$: In this case, we have:

$$\begin{array}{l} \Gamma_0' \vdash_{\mathtt{lin}} [t/x]s_0 : (\bigwedge\{\tau_1', \ldots, \tau_\ell'\} \to \rho, \eta) \Rightarrow p_0' : \times \boldsymbol{\gamma_1'} \to \cdots \times \boldsymbol{\gamma_k'} \to \mathtt{o} \dashv \Delta_0' \\ \Gamma_i' \vdash_{\mathtt{lin}} [t/x]s_1 : \tau_i' \Rightarrow P_i' : \times \boldsymbol{\gamma_i'} \dashv \Delta_i' \text{ for each } i \in \{1, \ldots, k\} \\ \Gamma_i' \vdash_{\mathtt{lin}} [t/x]s_1 : \tau_i' \Rightarrow p_{1,i}' \text{ for each } i \in \{k+1, \ldots, \ell\} \\ \tau = \begin{cases} (\rho, \eta) & \text{if } k = 0 \\ (\rho, \mathbf{nc}) & \text{if } k > 0 \end{cases} \\ \Gamma = \bigcup_{i \in \{0,1,\ldots,\ell\}} \Gamma_i' \qquad p = p_0' P_1' \cdots P_k' \qquad \tau_1' < \cdots < \tau_k' \\ \Delta = \biguplus_{i \in \{0,1,\ldots,k\}} \Delta_i' \end{array}$$

  By applying the induction hypothesis to $\Gamma_0' \vdash_{\mathtt{lin}} [t/x]s_0 : (\bigwedge\{\tau_1', \ldots, \tau_\ell'\} \to \rho, \eta) \Rightarrow p_0' : \times \boldsymbol{\gamma_1'} \to \cdots \times \boldsymbol{\gamma_k'} \to \mathtt{o} \dashv \Delta_0'$, we obtain:

$$\begin{array}{l} \Gamma_{0,0} \cup \{x : \tau_j \mid j \in I_0 \cup J_0\} \vdash_{\mathtt{lin}} s_0 : (\bigwedge\{\tau_1', \ldots, \tau_\ell'\} \to \rho, \eta) \\ \qquad \Rightarrow p_{0,0} : \times \boldsymbol{\gamma_1'} \to \cdots \times \boldsymbol{\gamma_k'} \to \mathtt{o} \dashv \Delta_{0,0} \uplus \{\boldsymbol{x_{\tau_j}}_0 : \boldsymbol{\gamma_{0,j}} \mid j \in I_0\} \\ \Gamma_{0,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow P_{0,j} : \times \boldsymbol{\gamma_{0,j}} \dashv \Delta_{0,j} \text{ and } \mathit{flag}(\tau_j) = \mathbf{nc} \text{ for } j \in I_0 \\ \Gamma_{0,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow p_{0,j} \text{ and } \mathit{flag}(\tau_j) \neq \mathbf{nc} \text{ for } j \in J_0 \\ p_0' = [P_{0,j}/\boldsymbol{x_{\tau_j}}_0]_{j \in I_0} p_{0,0} \\ \Gamma_0' \supseteq \bigcup_{j \in \{0\} \cup I_0 \cup J_0} \Gamma_{0,j} \qquad \Delta_0' = \uplus_{j \in \{0\} \cup I_0} \Delta_{0,j} \end{array}$$

  By applying the induction hypothesis to $\Gamma_i' \vdash_{\mathtt{lin}} [t/x]s_1 : \tau_i' \Rightarrow P_i' : \times \boldsymbol{\gamma_i'} \dashv \Delta_i'$ (where $i \in \{1, \ldots, k\}$), we have:

$$\begin{array}{l} \Gamma_{i,0} \cup \{x : \tau_j \mid j \in I_i \cup J_i\} \vdash_{\mathtt{lin}} s_1 : \tau_i' \Rightarrow P_{i,0} : \times \boldsymbol{\gamma_i'} \dashv \Delta_{i,0} \uplus \{\boldsymbol{x_{\tau_j}}_i : \boldsymbol{\gamma_{i,j}} \mid j \in I_i\} \\ \Gamma_{i,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow P_{i,j} : \times \boldsymbol{\gamma_{i,j}} \dashv \Delta_{i,j} \text{ and } \mathit{flag}(\tau_j) = \mathbf{nc} \text{ for } j \in I_i \\ \Gamma_{i,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow p_{i,j} \text{ and } \mathit{flag}(\tau_j) \neq \mathbf{nc} \text{ for } j \in J_i \\ P_i' = [P_{i,j}/\boldsymbol{x_{\tau_j}}_i]_{j \in I_0} U_{i,0} \\ \Gamma_i' \supseteq \bigcup_{j \in \{0\} \cup I_i \cup J_i} \Gamma_{i,j} \qquad \Delta_i' = \biguplus_{\{0\} \cup \in I_i} \Delta_{i,j} \end{array}$$

  By applying the induction hypothesis to $\Gamma_i' \vdash_{\mathtt{lin}} [t/x]s_1 : \tau_i' \Rightarrow p_{1,i}'$ (where $i \in \{k+1, \ldots, \ell\}$), we have:

$$\begin{array}{l} \Gamma_{i,0} \cup \{x : \tau_j \mid j \in I_i \cup J_i\} \vdash_{\mathtt{lin}} s_1 : \tau_i' \Rightarrow p_{i,0} \\ \Gamma_{i,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow P_{i,j} \text{ and } \mathit{flag}(\tau_j) = \mathbf{nc} \text{ for } j \in I_i \\ \Gamma_{i,j} \vdash_{\mathtt{lin}} t : \tau_j \Rightarrow p_{i,j} \text{ and } \mathit{flag}(\tau_j) \neq \mathbf{nc} \text{ for } j \in J_i \\ \Gamma_i \supseteq \Gamma_{i,0} \cup \bigcup_{j \in I_i \cup J_i} \Gamma_{i,j} \end{array}$$

By Lemma 15 and $\eta(\tau_i') \neq \mathbf{nc}$, we have $I_i = \emptyset$. Let

$$\begin{array}{l}
\Gamma_0 = \bigcup_{i\in\{0,\ldots,\ell\}} \Gamma_{i,0} \qquad \Delta_0 = \biguplus_{i\in\{0,\ldots,k\}} \Delta_{i,0} \\
I = \bigcup_{i\in\{0,\ldots,k\}} I_i \qquad J = \bigcup_{i\in\{0,\ldots,\ell\}} J_i \qquad p_0 = p_{0,0}P_{1,0}\cdots P_{k,0} \\
\Gamma_j = \bigcup_{i\in\{i\in\{0,\ldots,\ell\}\mid j\in I_i\}} \Gamma_{i,j} \qquad P_j = +_{i\in\{i\in\{0,\ldots,k\}\mid j\in I_i\}} P_{i,j} \\
\boldsymbol{x}_{\boldsymbol{\tau_j}} = +_{i\in\{i\in\{0,\ldots,k\}\mid j\in I_i\}} \boldsymbol{x}_{\boldsymbol{\tau_j}\boldsymbol{i}} \\
\boldsymbol{\gamma_j} = +_{i\in\{i\in\{0,\ldots,k\}\mid j\in I_i\}} \boldsymbol{\gamma_{i,j}} \qquad \Delta_j = \biguplus_{i\in\{0,\ldots,k\}} \Delta_{i,j} \text{ (for } j \in I)
\end{array}$$

For each $j \in J$, pick i such that $j \in J_i$ and let p_j and Γ_j be $p_{i,j}$ and $\Gamma_{i,j}$ respectively. Then we have the required conditions. □

We write $=_{\lambda,\mathtt{lin}}$ for the least equivalence relation including $\longrightarrow_{\lambda,\mathtt{lin}}$.

Lemma 17. *If $t \longrightarrow_{\mathcal{G}} t'$ and $\emptyset \vdash_{\mathtt{lin}} t' : \mathtt{o} \Rightarrow p' : \mathtt{o} \dashv \emptyset$, then there exists p such that $\emptyset \vdash_{\mathtt{lin}} t : \mathtt{o} \Rightarrow p : \mathtt{o} \dashv \emptyset$ with $p' =_{\lambda,\mathtt{lin}} p$ or $p' \longrightarrow_{\mathcal{G}',\mathtt{lin}} =_{\lambda,\mathtt{lin}} p$.*

Proof. This follows by induction on the derivation of $t_1 \longrightarrow_{\mathcal{G}} t_2$. Since the induction step is trivial, we show only the base case, where $t = A\,s_1 \cdots s_\ell$ and $t' = [s_1/x_1, \ldots, s_\ell/x_\ell]s$, with $A\,x_1 \cdots x_\ell \to s \in \mathcal{R}$. By Lemma 16 and $\emptyset \vdash_{\mathtt{lin}} t' : \mathtt{o} \Rightarrow p' : \mathtt{o} \dashv \emptyset$, we have:

$$\begin{array}{l}
\{x_k : \tau_{k,i} \mid k \in \{1,\ldots,\ell\}, i \in I_k \cup J_k\} \vdash_{\mathtt{lin}} s : \mathtt{o} \\
\qquad\qquad \Rightarrow p_0 : \mathtt{o} \dashv \{\boldsymbol{x_{k,i}} : \boldsymbol{\gamma_{k,i}} \mid k \in \{1,\ldots,\ell\}, i \in I_k\} \\
\mathit{flag}(\tau_{k,i}) = \mathbf{nc} \text{ for } k \in \{1,\ldots,\ell\}, i \in I_k \qquad \mathit{flag}(\tau_{k,i}) \neq \mathbf{nc} \text{ for } k \in \{1,\ldots,\ell\}, i \in J_k \\
\emptyset \vdash_{\mathtt{lin}} s_k : \tau_{k,i} \Rightarrow P_{k,i} : \times\boldsymbol{\gamma_{k,i}} \dashv \emptyset \text{ for } k \in \{1,\ldots,\ell\}, i \in I_k \\
\emptyset \vdash_{\mathtt{lin}} s_k : \tau_{k,i} \Rightarrow p_i \text{ for } k \in \{1,\ldots,\ell\}, i \in J_k \\
p' = [P_{k,i}/\boldsymbol{x_{k,\tau_{k,i}}}]_{k\in\{1,\ldots,\ell,i\in I_k\}} p_0
\end{array}$$

Let p_0' be a term such that $p_0 \downarrow_{\lambda,\mathtt{lin}} p_0'$. Let g be $\lambda\widetilde{\boldsymbol{x_{1,i}} : \boldsymbol{\gamma_{1,i}}_{i\in I_1}}.\cdots\lambda\widetilde{\boldsymbol{x_{\ell,i}} : \boldsymbol{\gamma_{\ell,i}}_{i\in I_\ell}}.p_0'$. If $\overline{g} \in \mathcal{C}$, then let p be

$$\langle g\rangle \widetilde{P_{1,i}}_{i\in I_1} \cdots \widetilde{P_{\ell,i}}_{i\in I_\ell}.$$

Then we have $p =_{\lambda,\mathtt{lin}} p'$ as required. Otherwise, let p be $A^g_\tau \widetilde{P_{1,i}}_{i\in I_1} \cdots \widetilde{P_{\ell,i}}_{i\in I_\ell}$. Here, $\tau = (\bigwedge\{\tau_{1,i} \mid i \in I_1 \cup J_1\} \to \cdots \to \bigwedge\{\tau_{\ell,i} \mid i \in I_\ell \cup J_\ell\} \to \mathtt{o}, \mathbf{nc})$. Then $p \longrightarrow_{\mathcal{G}',\mathtt{lin}} =_{\lambda,\mathtt{lin}} p'$ and $\emptyset \vdash t : \mathtt{o} \Rightarrow p : \mathtt{o} \dashv \emptyset$ hold as required. □

The following lemma guarantees that the reduction on linear extended terms of type $\mathtt{o}$ is confluent.

Lemma 18. *If $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup =_{\lambda,\mathtt{lin}})^*\pi$ and $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup =_{\lambda,\mathtt{lin}})^*\pi'$ for trees π and π', then $\pi = \pi'$.*

Proof. We can define the following trivial map $p^\flat$ from a linear extended term to the linear λ-calculus (with tuples).

$$\begin{array}{l}
x^\flat = x \qquad a^\flat = a \qquad (A^g)^\flat = g^\flat \\
(pP)^\flat = p^\flat P^\flat \qquad (\langle g\rangle P)^\flat = g^\flat P^\flat \\
(p_1,\ldots,p_k)^\flat = ({p_1}^\flat,\ldots,{p_k}^\flat) \qquad (\lambda(x_1,\ldots,x_k).g)^\flat = \lambda(x_1,\ldots,x_k).g^\flat.
\end{array}$$

Then it is trivial that $\longrightarrow_{\mathcal{G},\mathtt{lin}}$ and $\longrightarrow_{\lambda,\mathtt{lin}}$ preserve the β-equivalence of the image of $(\cdot)^\flat$. Thus, we have $p^\flat$ is β-equivalent to both π and π', which imply that $\pi = \pi'$. □

Proof of Theorem 5. Suppose $S \longrightarrow_{\mathcal{G}}^{*} \pi$. By repeated applications of Lemma 17 to $\emptyset \vdash_{\mathtt{lin}} \pi : \mathsf{o} \Rightarrow \pi$, we have $S_{\mathsf{o}}^{g}(\longrightarrow_{\mathcal{G}',\mathtt{lin}}=_{\lambda,\mathtt{lin}})^{*}\pi$. By induction on the number of non-terminals occurring in S_{o}^{g} (including those in annotations), we have $S_{\mathsf{o}} \Longrightarrow_{\mathcal{G}',\mathtt{lin}}^{*}=_{\lambda,\mathtt{lin}} \pi'$ for some tree π'. By Lemma 18, we have $\pi = \pi'$ as required. □

C Proofs for Section 4

C.1 Proof of Lemma 3

To show Lemma 3, we first evaluate the number of combinators in a linear extended term.

Lemma 19. *Suppose that p does not contain any consecutive applications of combinators. Then, the number of combinators in p is no greater than $(3|p| - 2)/4$.*

Proof. This follows by induction on the structure of p. Suppose p is of the form $h\, P_1 \cdots P_k$, where h is a terminal, a non-temrinal, or a combinator, and k may be 0 (in which case, h is not a combinator). if h is not a combinator, by the induction hypothesis, the number of combinators is bounded by

$$(3|P_1| - 2)/4 + \cdots + (3|P_k| - 2)/4 < (3|p| - 2)/4.$$

If h is a combinator, then $k \geq 1$ and (i) $P_1 = (p_1, \ldots, p_\ell)$ with $\ell > 1$, (ii) $P_1 = (h'\, P_1' \cdots P_\ell')$ with h' is not a combinator, or (iii) $P_1 = (h'\, P_1' \cdots P_\ell')$ with h' is a combinator and $\ell > 1$, In case (i), the number of combinators is bounded by

$$\begin{aligned}
&1 + \Sigma_{i\in\{1,\ldots,\ell\}}(3|p_i| - 2)/4 + \Sigma_{j\in\{2,\ldots,k\}}(3|P_j| - 2)/4\\
&= (3(1 + \Sigma_{i\in\{1,\ldots,\ell\}}|p_i| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2\ell - 2(k-1) + 1)/4\\
&< (3(1 + \Sigma_{i\in\{1,\ldots,\ell\}}|p_i| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2)/4 = (3|p| - 2)/4
\end{aligned}$$

as required. In case (ii), the number of combinators is bounded by

$$\begin{aligned}
&1 + \Sigma_{i\in\{1,\ldots,\ell\}}(3|P_i'| - 2)/4 + \Sigma_{j\in\{2,\ldots,k\}}(3|P_j| - 2)/4\\
&= (3(2 + \Sigma_{i\in\{1,\ldots,\ell\}}|P_i'| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2\ell - 2(k-1) - 2)/4\\
&\leq (3(2 + \Sigma_{i\in\{1,\ldots,\ell\}}|P_i'| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2)/4 = (3|p| - 2)/4
\end{aligned}$$

as required. Finally, in case (iii), the number of combinators is bounded by

$$\begin{aligned}
&2 + \Sigma_{i\in\{1,\ldots,\ell\}}(3|P_i'| - 2)/4 + \Sigma_{j\in\{2,\ldots,k\}}(3|P_j| - 2)/4\\
&= (3(2 + \Sigma_{i\in\{1,\ldots,\ell\}}|P_i'| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2\ell - 2(k-1) + 2)/4\\
&\leq (3(2 + \Sigma_{i\in\{1,\ldots,\ell\}}|P_i'| + \Sigma_{j\in\{2,\ldots,k\}}|P_j|) - 2)/4 = (3|p| - 2)/4
\end{aligned}$$

as required. □

Lemma 20. *Suppose that p does not contain more than two consecutive applications of combinators. Then, the number of combinators in p is no greater than $6|p|/7$.*

Proof. Suppose that the number of two consecutive applications of combinators in p is x and the number of combinators in p is y. Then $2x \leq y$. Let p' be the linear extended term obtained by replacing each two consecutive applications of combinators by an application of a single combinator. By the constuction, we have $|p'| = |p| - x$, and the number of combinators in p' is $y - x$. By Lemma 20, we have $y - x \leq (3|p'| - 2)/4 \leq 3(|p| - x)/4$. Thus, we have $y \leq 3|p|/4 + x/4 \leq 3|p|/4 + y/8$, from which we obtain $y \leq (3/4 \times 8/7)|p| = 6|p|/7$. □

We are now ready to prove Lemma 3.

Proof of Lemma 3. Suppose $p{\downarrow_{\lambda,\mathtt{lin}}}p'$ and p does not contain more than two consecutive applications of combinators. By Lemma 20, the number of combinators in p is bounded by $6|p|/7$. Therefore, the number of terminals or non-terminals in p is at least $|p|/7$. Since $\longrightarrow_{\lambda,\mathtt{lin}}$ does not decrease the number of terminals or non-terminals, $|p'| \geq |p|/7$. Thus, we have $|p| \leq c_1|p'|$ for $c_1 = 7$. □

C.2 Proof of Lemma 5

We first extend the notion of extended permutators (which were defined in Section 4 only for λ-terms without tuples) to terms with tuples.

Definition 8. *A linear λ-term is an **extended permutator** if it is in $\beta\eta$ long normal form and of the form*

$$\lambda(y_{1,1}, \ldots, y_{1,k_1}). \cdots \lambda(y_{i,1}, \ldots, y_{i,k_i}). \lambda x. \lambda(y_{i+1,1}, \ldots, y_{i+1,k_i}). \cdots \lambda(y_{\ell,1}, \ldots, y_{\ell,k_\ell}).$$
$$x\,(z_{1,1}, \ldots, z_{1,k'_1}) \cdots (z_{\ell',1}, \ldots, z_{1,k'_{\ell'}})$$

where $z_{1,1}, \ldots, z_{1,k'_1}, \ldots, z_{\ell',1}, \ldots, z_{1,k'_{\ell'}}$ is a permutation of $y_{1,1}, \ldots, y_{1,k_1}, \ldots, y_{\ell,1}, \ldots, y_{\ell,k_\ell}$.

In other words, a linear λ-term p is an extended permutator if $\overline{p} \in \mathcal{C}$.

Lemma 21. *If $\Delta \vdash p : \gamma \Rightarrow u \dashv \Delta'$, then $\Delta' \vdash_{\mathtt{L}} u : \gamma$.*

Proof. This follows by straightforward induction on the derivation of $\Delta \vdash p : \gamma \Rightarrow u \dashv \Delta'$. □

Proof of Lemma 5. Suppose $\emptyset \vdash p : \mathtt{o}$ and $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^* \pi$. By straightforward induction on the derivation of $\emptyset \vdash p : \mathtt{o}$, we have u such that $\emptyset \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta$ and $\Delta \vdash u : \mathtt{o}$. Since $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^* \pi$ and a reductions by $\longrightarrow_{\mathcal{G},\mathtt{lin}}$ or $\longrightarrow_{\lambda,\mathtt{lin}}$ never deletes or copies terminal symbols, u contains the same number of occurrences of each terminal symbol as π. By the definition of the transformation relation $\emptyset \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta$, 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 p, hence also the same as the number of occurrences of a in π. □

C.3 Proof of Lemma 6

Lemma 22. *There is no linear extended term p such that $\emptyset \vdash p : \mathtt{o} \Rightarrow u : \mathtt{o} \dashv \emptyset$.*

Proof. By Lemma 21, we have $\emptyset \vdash_{\mathtt{L}} u : \mathtt{o}$. But then we must have $\emptyset \vdash_{\mathtt{L}} u' : \mathtt{o}$ for the β-normal form u' of u. This is impossible. (Recall that we do not have a constant in the linear λ-calculus.) □

Lemma 23. *Suppose that $\Delta \vdash p{:}\gamma \Rightarrow u \dashv \Delta'$ with $\mathtt{order}(\gamma) = 1$ and $\mathit{codom}(\Delta') \subseteq \{\mathtt{o}\}$. Suppose also that $p{\downarrow_{\mathtt{lin}}}$, then $\gamma = \mathtt{o} \to \mathtt{o}$, $\Delta = \Delta' = \emptyset$, and the derivation for $\emptyset \vdash p : \gamma \Rightarrow u \dashv \emptyset$ must contain a judgment of the form $\emptyset \vdash F : \gamma' \Rightarrow u_0 \dashv \emptyset$ such that $u_0 \longrightarrow^* u'_0$ for some extended permutator u'_0.*

Proof. By Lemma 21, we have $\Delta \vdash_{\mathtt{L}} u : \gamma$. Since γ is an order-1 type, the $\beta\eta$ long normal form of u must be $\lambda x.x$. This implies $\gamma = \mathtt{o} \to \mathtt{o}$, and $\Delta = \Delta' = \emptyset$. We show the remaining condition by induction on the structure of p. Since $\Delta = \Delta' = \emptyset$, p must be of the form $\langle f \rangle\, P_1 \cdots P_\ell$ (with $\ell > 0$) or $A\, P_1 \cdots P_\ell$ (where ℓ may be 0). If $\ell > 0$ (in both cases), by Lemma 22, the order of P_1 must be 1 (because the image of the translation is well-typed under the empty type environment). By the induction hypothesis, the derivation for the transformation of P_1 must contain a judgment of the required form. If $\ell = 0$, then $p = A$. Then A is transformed to u, which is reduced to the extended permutator $\lambda x.x$ as required. □

Lemma 24. *Let p be a linear extended term that does not contain any terminal symbols. If $\Delta \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta'$ with $\mathit{codom}(\Delta') \subseteq \{\mathtt{o}\}$, and if $p{\downarrow_{\mathtt{lin}}}$, then $\Delta = \Delta' = x^{(i)} : \mathtt{o}$. Furthermore, either $p = x^{(i)}$, or the derivation for $\Delta \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta'$ must contain a judgment of the form $\emptyset \vdash F : \gamma' \Rightarrow u_0 \dashv \emptyset$ such that $u_0 \longrightarrow^* u'_0$ for some extended permutator u'_0.*

Proof. By Lemma 21, we have $\Delta' \vdash_{\mathtt{L}} u : \mathtt{o}$. Thus, the normal form of u must be a variable, and $\Delta' = x^{(i)} : \mathtt{o}$. This also implies $\Delta = x^{(i)} : \mathtt{o}$. We show the remaining condition by induction on the structure of p. Suppose that p is not a variable. Then p must be of the form $A\, P_1 \cdots P_k$. Because $\Delta \neq \emptyset$, it must be the case that $k > 0$. If the order of P_i is 1 for some $i \leq k$, then by Lemma 23, the derivation for the transformation of P_i contains a judgment of the required condition. Otherwise, every P_i has order 0. If none of $P_1, \ldots, P_k$ contains a judgment of the required form, then by the induction hypothesis, it must be the case that $k = 1$ and $P_1 = (x^{(i)})$ (because Δ contains a single variable). In this case, $u = u_0\, x^{(i)}$, with $\emptyset \vdash F : \mathtt{o} \to \mathtt{o} \Rightarrow u_0 \dashv \emptyset$. Since $u \longrightarrow^* x^{(i)}$, the normal form of u_0 must be $\lambda x : \mathtt{o}.x$. Thus, the judgment $\emptyset \vdash F : \mathtt{o} \to \mathtt{o} \Rightarrow u_0 \dashv \emptyset$ satisfies the required condition. □.

The following is the key lemma, which states that a non-terminal cannot be transformed to an extended permutator.

Lemma 25. *Let $\mathcal{G}$ be an order-2 extended grammar over $\mathcal{C}_m$ with $\mathtt{ar}(\mathcal{G}) \leq m$ and A be a non-terminal of $\mathcal{G}$. There is no extended permutator u such that $\emptyset \vdash A : \gamma \Rightarrow u' \dashv \Delta$ with $u' \longrightarrow^* u$.*

Proof. The proof proceeds by the induction on the number of applications of rule LX-NT for deriving $\emptyset \vdash A : \gamma \Rightarrow u' \dashv \Delta$. Suppose $\emptyset \vdash A : \gamma \Rightarrow u' \dashv \Delta$ with $u' \longrightarrow^* u$ for an extended permutator u. By Lemma 21 and the subject reduction property for the simply-typed linear λ-calculus, we have $\Delta \vdash_{\mathtt{L}} u : \gamma$. Thus, it must be the case that $\Delta = \emptyset$.

We have

$$
\begin{array}{l}
A\, x_1 \ldots x_k \to \overline{p_0} \in \mathcal{R} \\
\boldsymbol{x_1} : \boldsymbol{\gamma_1}, \ldots, \boldsymbol{x_k} : \boldsymbol{\gamma_k} \vdash p_0 : \mathtt{o} \Rightarrow u_0 \dashv \boldsymbol{x_1} : \boldsymbol{\gamma_1}, \ldots, \boldsymbol{x_k} : \boldsymbol{\gamma_k} \\
\gamma = (\times \boldsymbol{\gamma_1}) \to \cdots \to (\times \boldsymbol{\gamma_k}) \to \mathtt{o} \\
u' = \lambda \boldsymbol{x_1}. \cdots \lambda \boldsymbol{x_k}. u_0 \\
u_0 \longrightarrow^* x_j^{(i)}\, U_1 \cdots U_h \\
\{x_j^{(i)}\} \uplus \widehat{U_1} \uplus \cdots \uplus \widehat{U_h} = \widehat{\boldsymbol{x_1}} \uplus \cdots \uplus \widehat{\boldsymbol{x_k}}.
\end{array}
$$

Here, $\widehat{U}$ denotes $\{u_1, \ldots, u_k\}$ for $U = (u_1, \ldots, u_k)$. Since $\emptyset \vdash A : \gamma \Rightarrow u' \dashv \emptyset$, p_0 must be either of the form $x_i\, P_1 \cdots P_\ell$ or $A'\, P_1 \cdots P_\ell$. In the former case, if $\ell = 0$, then $k = 1$ with $p_0 = x_1$, which contradicts with the assumption that the extended grammar being considered is an image of the translation. Otherwise (i.e. if $\ell > 0$), the order of P_i must be 0 (because the order of variable x_i is at most 1). By Lemma 24 and the induction hypothesis, every element of P_i must be a variable. However, this implies $\lambda x_1. \cdots \lambda x_k. \overline{p_0} \in \mathcal{C}_m$, which violates the condition that $\mathcal{G}$ is an extended grammar over $\mathcal{C}_m$.

In the latter case (where $p_0 = A'\, P_1 \cdots P_\ell$), u_0 must be of the form $u_0'\, U_1' \ldots U_\ell'$, where u_0' is the image of the transformation of A'. By Lemma 23 and the induction hypothesis, at most one element among the elements of P_i' is order 1, and the other elements are order 0, since an order-1 term must contain a free variable of order-1, but there is only one order-1 free variable. Thus, the type of u_0' is $\gamma_{1,1} \times \cdots \gamma_{1,k_1} \to \cdots \to \gamma_{\ell,1} \times \cdots \gamma_{\ell,k_\ell} \to \mathtt{o}$ with $\gamma_{i,j}$ being order 1 for only at most one (i, j) and $\gamma_{i,j} = \mathtt{o}$ for the other (i, j). This implies that the normal form of u_0' must be an extended permutator (note that u_0' must be a closed linear λ-term). This is however impossible by the induction hypothesis. □

Proof of Lemma 6. By Lemma 5, there exists u such that $\emptyset \vdash p : \mathtt{o} \Rightarrow u \dashv \Delta$ and Δ satisfies the second condition. By the definition of the transformation relation, $\mathit{asize}(u) \geq |p|$. Let M be the pure linear λ-term obtained from u by applying the currying transformation and then normalizing every top-level λ-abstractions (but leaving top-level redexes as they are). Then M satisfies the required conditions. To check the last condition, suppose M contains a subterm $(L_1(L_2 M'))$ for permutators L_1 and L_2. Then by Lemma 25, the corresponding subterm of p must be of the form $\langle g_1 \rangle (p_1, \ldots, p_k)$ with $p_1 = \langle g_2 \rangle P$. By the condition $p{\downarrow_{\mathtt{lin}}}$, it must be the case that $k > 1$. Therefore, by the definition of extended permutators, $p_1 = \langle g_2 \rangle P$ must have type $\mathtt{o}$, which contradicts with the condition $p{\downarrow_{\mathtt{lin}}}$. □

C.4 Proof of Lemma 4

We show that for every order-2 pure λ-term M that satisfies all the conditions of Lemma 6, $\mathit{asize}(M)$ is linearly bounded by the size of π (see Theorem 8). By combining it with Lemma 6, we obtain Lemma 4.

Below we consider the type judgment $\Delta \vdash_{\mathtt{L}} u : \gamma$ of the linear type system, restricted to the case where u is a pure linear term (i.e., all the tuples have size 1); thus we usually write $\mathcal{K} \vdash_{\mathtt{L}} M : \kappa$ for the judgment.

In Theorem 6, we have stated that *extended* permutators occur in restricted positions of M. To give the bound for $\mathit{asize}(M)$, it is actually sufficient to assume that standard permutators [2] occur in the restricted positions.

Definition 9 (permutators). *A pure linear λ-term M is called a* ***permutator*** *if M is one of the following forms:*

(i) $\lambda x : \mathtt{o}.x$
(ii) $\lambda y : \underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{k} \to \mathtt{o}.\lambda x_1 : \mathtt{o}. \cdots \lambda x_k : \mathtt{o}.y\, x_{\theta(1)} \cdots x_{\theta(k)}$, *where θ is a permutation on $\{1, \ldots, k\}$.*

Obviously, the set of permutators is a subset of the set of extended permutators.

We define $\mathbf{w}(\mathcal{K})$ as the number of bindings of the form $x : \mathtt{o}$ in $\mathcal{K}$, i.e.,

$$\mathbf{w}(\mathcal{K}) = |\{x \mid x : \mathtt{o} \in \mathcal{K}\}|.$$

We associate two costs $\mathbf{wp}(\kappa)$ and $\mathbf{wc}(\kappa)$ for each type. Intuitively,

- $\mathbf{wp}(\kappa)$ is the cost for *producing* a term of type κ, i.e., the minimal $\mathbf{w}(\mathcal{K})$ such that $\mathit{codom}(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$ and $\mathcal{K} \vdash_{\mathtt{L}} M : \kappa$ for some linear term M (that does not contain extended permutators as subterms).
- $\mathbf{wc}(\kappa)$ is the cost for *consuming* a term of type κ, i.e., the minimal $\mathbf{w}(\mathcal{K})$ such that $\mathit{codom}(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$ and $\mathcal{K}, x{:}\kappa \vdash_{\mathtt{L}} C[x] : \mathtt{o}$ for some linear applicative context C (that does not contain permutators as subterms). Here, an applicative context is a context generated by:

$$C ::= [\,] \mid M\, C \mid C\, M.$$

The costs $\mathbf{wp}(\kappa)$ and $\mathbf{wc}(\kappa)$ are formally defined by:

$$\mathbf{wp}((\mathtt{o}^{k_1} \to \mathtt{o}) \to \cdots \to (\mathtt{o}^{k_\ell} \to \mathtt{o}) \to \mathtt{o}) = \begin{cases} 1 & \text{if } k_1 = \ell - 1 \text{ and } k_i = 0 \text{ for every } i \in \{2, \ldots, \ell\} \\ \max(0, (k_1 + \cdots + k_\ell) - \ell + 1) & \text{otherwise} \end{cases}$$

$$\mathbf{wc}((\mathtt{o}^{k_1} \to \mathtt{o}) \to \cdots \to (\mathtt{o}^{k_\ell} \to \mathtt{o}) \to \mathtt{o}) = \mathbf{wp}(\mathtt{o}^{k_1} \to \mathtt{o}) + \cdots + \mathbf{wp}(\mathtt{o}^{k_\ell} \to \mathtt{o}) = |\{i \in \{1, \ldots, \ell\} \mid k_i \leq 1\}|$$

Here, $\mathtt{o}^k \to \kappa$ is a shorthand for $\underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{k} \to \mathtt{o}$.

Remark 1. As a special case, we have $\mathbf{wc}(\mathsf{o}^k \to \mathsf{o}) = k$ and

$$\mathbf{wp}(\mathsf{o}^k \to \mathsf{o}) = \begin{cases} 1 \text{ if } k \leq 1 \\ 0 \text{ if } k > 1 \end{cases}$$

The property stated in the following lemma is essentially the standard one on the relationship between the numbers of nodes and leaves in a tree.

Lemma 26. *If* $x_1 : (\mathsf{o}^{k_1} \to \mathsf{o}), \ldots, x_n : (\mathsf{o}^{k_n} \to \mathsf{o}) \vdash_{\mathtt{L}} M : \mathsf{o}$*, then* $1 + \sum_{i=1}^{n}(k_i - 1) = 0$.

Proof. This follows by induction on the structure of M. By subject reduction, we can assume that M is in β-normal form. Since M has type o, M must be either a variable or an application.

- Case where M is a variable. In this case, it must be the case that $n = 1$ and $k_1 = 0$ with $M = x_1$. The result follows immediately.
- Case where M is an application. In this case, M must be of the form $x_\ell\, M_1 \cdots M_{k_i}$, with $\mathcal{K}_j \vdash_{\mathtt{L}} M_j : \mathsf{o}$ for each $j \in \{1, \ldots, k_i\}$ and $\{x_\ell : \mathsf{o}^{k_\ell} \to \mathsf{o}\} \uplus (\biguplus_{j \in \{1,\ldots,k_i\}} \mathcal{K}_j) = \{x_1 : (\mathsf{o}^{k_1} \to \mathsf{o}), \ldots, x_n : (\mathsf{o}^{k_n} \to \mathsf{o})\}$. Let $I_j = \{i \mid x_i \in \mathit{dom}(\mathcal{K}_j)\}$. By the induction hypothesis, $1 + \sum_{i \in I_j}(k_i - 1) = 0$. Therefore, we have

  $$\begin{array}{l} 1 + \sum_{i=1}^{n}(k_i - 1) = 1 + (k_\ell - 1) + \sum_{j=1}^{k_\ell} \sum_{i \in I_j}(k_i - 1) \\ = k_\ell + \sum_{j=1}^{k_\ell}(-1) \\ = k_\ell - k_\ell = 0 \end{array}$$

  as required. □

Lemma 27. *If* $x_1 : \mathsf{o}, \ldots, x_m : \mathsf{o}, y_1 : \mathsf{o} \to \mathsf{o} \to \mathsf{o}, \ldots, y_n : \mathsf{o} \to \mathsf{o} \to \mathsf{o} \vdash_{\mathtt{L}} M : \mathsf{o}$ *and* M *is a* β*-normal form, then* $n = m - 1$.

Proof. This is a special case of Lemma 26. □

Lemma 28. *Suppose that* M *is in* $\beta\eta$ *long normal form and* M *is not a permutator. If* $\mathcal{K} \vdash_{\mathtt{L}} M : \kappa$ *and* $\mathit{codom}(\mathcal{K}) \subseteq \{\mathsf{o}, \mathsf{o} \to \mathsf{o} \to \mathsf{o}\}$*, then* $\mathbf{w}(\mathcal{K}) \geq \mathbf{wp}(\kappa)$.

Proof. Let $\kappa = (\mathsf{o}^{k_1} \to \mathsf{o}) \to \cdots \to (\mathsf{o}^{k_n} \to \mathsf{o}) \to \mathsf{o}$. If $k_1 = \ell - 1$ and $k_i = 0$ for every $i \in \{2, \ldots, \ell\}$, then the only closed term that is in $\beta\eta$ long normal form is a permutator. Since M is not a permutator, it must be the case that $\mathbf{w}(\mathcal{K}) \geq 1 = \mathbf{wp}(\kappa)$. Otherwise, $M = \lambda x_1 \cdots x_n . M'$ with $\mathcal{K}, x_1 : (\mathsf{o}^{k_1} \to \mathsf{o}), \ldots, (\mathsf{o}^{k_n} \to \mathsf{o}) \vdash_{\mathtt{L}} M' : \mathsf{o}$. If By Lemma 26, we have $1 - \mathbf{w}(\mathcal{K}) + \sum_{i=1}^{n}(k_i - 1) \leq 0$, so, we have

$$\mathbf{w}(\mathcal{K}) \geq 1 + \sum_{i=1}^{n}(k_i - 1) = k_1 + \cdots + k_n - n + 1.$$

Since $\mathbf{w}(\mathcal{K}) \geq 0$, we have

$$\mathbf{w}(\mathcal{K}) \geq \max(0, 1 + \sum_{i=1}^{n}(k_i - 1) = k_1 + \cdots + k_n - n + 1) = \mathbf{wp}(\kappa)$$

as required. □

Lemma 29. $\mathbf{wc}(\kappa_1)+\cdots+\mathbf{wc}(\kappa_n) \leq \mathbf{wp}(\kappa_1 \to \cdots \to \kappa_n \to \kappa)+\mathbf{wc}(\kappa)+n-1$ *holds for all* $n \geq 0$, $\kappa_1, \ldots, \kappa_n$ *and* κ *(such that* $\mathtt{order}(\kappa_i) \leq 1$ *and* $\mathtt{order}(\kappa) \leq 2$*). If* $\mathbf{wp}(\kappa \to \kappa') = 0$, *then* $\mathbf{wc}(\kappa) < \mathbf{wc}(\kappa')$ *holds.*

Proof. Without loss of generality, we can assume:

$$\kappa_i = \mathtt{o}^{k_i} \to \mathtt{o} \qquad \kappa = (\mathtt{o}^{k_{n+1}} \to \mathtt{o}) \to \cdots \to (\mathtt{o}^{k_{n+m}} \to \mathtt{o}) \to \mathtt{o}$$

By the definition of **wp** and **wc**, we have:

$$\begin{aligned}
&\mathbf{wp}(\kappa_1 \to \cdots \to \kappa_n \to \kappa) + \mathbf{wc}(\kappa) + n - 1 - (\mathbf{wc}(\kappa_1) + \cdots + \mathbf{wc}(\kappa_n)) \\
&\geq k_1 + \cdots + k_{n+m} - (n+m) + 1 + |\{i \in \{n+1, \ldots, n+m\} \mid k_i \leq 1\}| + n - 1 - (k_1 + \cdots + k_n) \\
&= (k_{n+1} + \cdots + k_{n+m} - m) + |\{i \in \{n+1, \ldots, n+m\} \mid k_i \leq 1\}| \\
&= (k_{n+1} - 1) + \cdots + (k_{n+m} - 1) + |\{i \in \{n+1, \ldots, n+m\} \mid k_i \leq 1\}| \\
&\geq -|\{i \in \{n+1, \ldots, n+m\} \mid k_i = 0\}| + |\{i \in \{n+1, \ldots, n+m\} \mid k_i \leq 1\}| \\
&\geq 0
\end{aligned}$$

Thus, we have $\mathbf{wc}(\kappa_1)+\cdots+\mathbf{wc}(\kappa_n) \leq \mathbf{wp}(\kappa_1 \to \cdots \to \kappa_n \to \kappa)+\mathbf{wc}(\kappa)+n-1$ as required.

To show the second property of the lemma, suppose that $\kappa = \mathtt{o}^{k_1} \to \mathtt{o}$ and $\kappa' = (\mathtt{o}^{k_2} \to \mathtt{o}) \to \cdots \to (\mathtt{o}^{k_m} \to \mathtt{o}) \to \mathtt{o}$ with $\mathbf{wp}(\kappa \to \kappa') = 0$. We need to show $\mathbf{wc}(\kappa') - \mathbf{wc}(\kappa) = |\{i \in \{2, \ldots, m\} \mid k_i \leq 1\}| - k_1 > 0$.

By the assumption $\mathbf{wp}(\kappa \to \kappa') = 0$, we have $k_1+\cdots+k_m-m+1 \leq 0$, which implies $k_1 \leq (1-k_2)+\cdots+(1-k_m)$. Thus, $|\{j \in \{2, \ldots, m\} \mid k_j = 0\}| \geq k_1$. If it were the case $m-1 = k_1$, then $k_2 = \cdots = k_m = 0$ and $\kappa = \kappa' = \mathtt{o}^{k_1} \to \mathtt{o}$, so we have $\mathbf{wp}(\kappa \to \kappa') = 1$, which contradicts with the assumption $\mathbf{wp}(\kappa \to \kappa') = 0$. Thus, we have $m - 1 > k_1$. By $k_1 + \cdots + k_m - m + 1 \leq 0$, we have:

$$\begin{aligned}
&|\{i \in \{2, \ldots, m\} \mid k_j \leq 1\}| - k_1 = (m - 1 - |\{i \in \{2, \ldots, m\} \mid k_j \geq 2\}|) - k_1 \\
&\geq m - 1 - (k_2 + \cdots + k_m)/2 - k_1 \\
&\geq m - 1 - (k_2 + \cdots + k_m)/2 + (k_2 + \cdots + k_m) - m + 1 \\
&= (k_2 + \cdots + k_m)/2
\end{aligned}$$

If $(k_2+\cdots+k_m)/2 > 0$, then the result follows immediately. If $(k_2+\cdots+k_m)/2 = 0$, then from the second line of the above inequalities, we have

$$\begin{aligned}
&|\{i \in \{2, \ldots, m\} \mid k_j \leq 1\}| - k_1 \\
&\geq m - 1 - (k_2 + \cdots + k_m)/2 - k_1 = m - 1 - k_1 > 0
\end{aligned}$$

□

Lemma 30. *Suppose that* M *is an applicative combination of pure linear* λ*-terms in* $\beta\eta$ *long normal form, and that* M *does not contain permutators. If* $\mathcal{K} \vdash_{\mathtt{L}} M : \kappa$ *and* $\mathit{codom}(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$ *with* $\mathtt{order}(\kappa) \leq 1$, *then* $\mathit{asize}(M) \leq 7\mathbf{w}(\mathcal{K}) + 4\mathbf{wc}(\kappa) - 6$.

Proof. This follows by induction on the structure of M.

– Case $M = x$: In this case, $\mathcal{K} = x : \kappa$. Since $\mathit{codom}(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$, κ is either $\mathtt{o}$ or $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$. In the former case, $\mathbf{w}(\mathcal{K}) = 1$ and $\mathbf{wc}(\kappa) = 0$, so that $7\mathbf{w}(\mathcal{K}) + 4\mathbf{wc}(\kappa) - 6 = 1 = \mathit{asize}(M)$. In the latter case, $\mathbf{w}(\mathcal{K}) = 0$ and $\mathbf{wc}(\kappa) = 2$, so that $7\mathbf{w}(\mathcal{K}) + 4\mathbf{wc}(\kappa) - 6 = 2 \geq \mathit{asize}(M)$.
– Case M is a λ-abstraction: By the assumption on the form of M, M is $\beta\eta$ long normal form $\lambda x_1 \cdots x_k.M'$ with $\mathcal{K}, x_1 : \mathtt{o}, \ldots, x_k : \mathtt{o} \vdash_{\mathtt{L}} M' : \mathtt{o}$ and $\kappa = \mathtt{o}^k \to \mathtt{o}$. The result follows immediately if $\mathbf{w}(\mathcal{K}) > 0$ or $\mathbf{wc}(\kappa) = k > 1$ holds. We show that indeed $\mathbf{w}(\mathcal{K}) > 0$ or $\mathbf{wc}(\kappa) = k > 1$ must be the case. Suppose $\mathbf{w}(\mathcal{K}) = 0$ and $k \leq 1$. Since $\mathbf{w}(\mathcal{K}) = 0$, $\mathcal{K}$ must be of the form $y_1 : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \ldots, y_m : \mathtt{o} \to \mathtt{o} \to \mathtt{o}$. By Lemma 27, $m = k - 1$, which implies $m = 0$ and $k = 1$. Thus, we have $x : \mathtt{o} \vdash_{\mathtt{L}} M' : \mathtt{o}$. By the typing rules, M' must be x, but then $M = \lambda x.x$, which contradicts with the requirement that M must not be a permutator.
– Case $M = L\, M_1 \ \ldots \ M_\ell$, with $\ell > 0$ and L is not an application. In this case, we have:

$$\begin{array}{l} \mathcal{K}_0 \vdash_{\mathtt{L}} L : \kappa_1 \to \cdots \to \kappa_\ell \to \kappa \\ \mathcal{K}_i \vdash_{\mathtt{L}} M_i : \kappa_i \\ \mathcal{K} = \mathcal{K}_0 \uplus \cdots \uplus \mathcal{K}_\ell \\ \mathtt{order}(\kappa_i) \leq 1 \end{array}$$

Since $\mathtt{order}(\kappa_i) \leq 1$, by the induction hypothesis, we have

$$\mathit{asize}(M_i) \leq 7\mathbf{w}(\mathcal{K}_i) + 4\mathbf{wc}(\kappa_i) - 6$$

Thus,

$$\begin{array}{l} \mathit{asize}(M) = 1 + \mathit{asize}(M_1) + \cdots + \mathit{asize}(M_\ell) \\ \leq 1 + 7(\mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4(\mathbf{wc}(\kappa_1) + \cdots + \mathbf{wc}(\kappa_\ell)) - 6\ell \end{array}$$

If $\ell \geq 2$, then by Lemmas 28 and 29, we have

$$\begin{array}{ll} \mathit{asize}(M) \leq 1 + 7(\mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4(\mathbf{wc}(\kappa_1) + \cdots + \mathbf{wc}(\kappa_\ell)) - 6\ell & \\ \leq 7(\mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4(\mathbf{wp}(\kappa_1 \to \cdots \to \kappa_\ell \to \kappa) + \mathbf{wc}(\kappa) + \ell - 1) - (6\ell - 1) & \\ \qquad \text{(by Lemma 29)} & \\ = 7(\mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4\mathbf{wp}(\kappa_1 \to \cdots \to \kappa_\ell \to \kappa) + 4\mathbf{wc}(\kappa) - (2\ell + 3) & \\ \leq 7(\mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4\mathbf{w}(\mathcal{K}_0) + 4\mathbf{wc}(\kappa) - 6 & \text{(by Lemma 28)} \\ \leq 7(\mathbf{w}(\mathcal{K}_0) + \mathbf{w}(\mathcal{K}_1) + \cdots + \mathbf{w}(\mathcal{K}_\ell)) + 4\mathbf{wc}(\kappa) - 6 & \\ = 7\mathbf{w}(\mathcal{K}) + 4\mathbf{wc}(\kappa) - 6 & (\text{by } \mathcal{K} = \mathcal{K}_0 \uplus \cdots \uplus \mathcal{K}_\ell) \end{array}$$

If $\ell = 1$ and $\mathbf{wp}(\kappa_1 \to \kappa) \geq 1$, then we have

$$\begin{array}{ll} \mathit{asize}(M) \leq 1 + 7\mathbf{w}(\mathcal{K}_1) + 4\mathbf{wc}(\kappa_1) - 6 & \\ \leq 1 + 7\mathbf{w}(\mathcal{K}_1) + 4(\mathbf{wp}(\kappa_1 \to \kappa) + \mathbf{wc}(\kappa)) - 6 & \text{(by Lemma 29)} \\ \leq 7\mathbf{w}(\mathcal{K}_1) + 5\mathbf{wp}(\kappa_1 \to \kappa) + 4\mathbf{wc}(\kappa) - 6 & (\text{by } \mathbf{wp}(\kappa_1 \to \kappa) \geq 1) \\ \leq 7\mathbf{w}(\mathcal{K}_1) + 5\mathbf{w}(\mathcal{K}_0) + 4\mathbf{wc}(\kappa) - 6 & \text{(by Lemma 28)} \\ \leq 7(\mathbf{w}(\mathcal{K}_1) + \mathbf{w}(\mathcal{K}_0)) + 4\mathbf{wc}(\kappa) - 6 & \\ = 7\mathbf{w}(\mathcal{K}) + 4\mathbf{wc}(\kappa) - 6 & (\text{by } \mathcal{K} = \mathcal{K}_0 \uplus \mathcal{K}_1) \end{array}$$

If $\ell = 1$ and $\mathbf{wp}(\kappa_1 \to \kappa) = 0$, then we have

$$\begin{array}{l} \mathit{asize}(M) \leq 1 + 7\mathbf{w}(\mathcal{K}_1) + 4\mathbf{wc}(\kappa_1) - 6 \\ < 7\mathbf{w}(\mathcal{K}_1) + 4(\mathbf{wc}(\kappa_1) + 1) - 6 \\ \leq 7\mathbf{w}(\mathcal{K}_1) + 4\mathbf{wc}(\kappa) - 6 \qquad \text{(by the second claim of Lemma 29)} \end{array}$$

as required. □

Theorem 6. *Suppose that M is an applicative combination of pure linear λ-terms each of which is in $\beta\eta$ long normal form, and that M does not contain permutators. If $\mathcal{K} \vdash_{\mathtt{L}} M : \mathtt{o}$ and $\mathit{codom}(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$, then $\mathit{asize}(M) \leq 7\mathbf{w}(\mathcal{K}) - 6$.*

Proof. This follows as an immediate corollary of Lemma 30. □

The following lemma is similar to Lemma 19.

Lemma 31. *Let M be an applicative combination of pure linear λ-terms each of which is in $\beta\eta$ long normal form (that may include permutators). Suppose that M itself is not a permutator, and that M contains neither (i) a consecutive application of permutators (i.e. a subterm of the form $L_1(L_2\, N)$ where L_1 and L_2 are permutators) nor (ii) a permutator in an argument position (i.e., a subterm of the form $N\, L$ where L is a permutator). Then the number of permutators in M is no greater than $(3\mathit{asize}(M) - 2)/4$.*

Proof. This follows by induction on the structure of M. Suppose that M is of the form $X\, M_1 \cdots M_k$ where X is a variable or an abstraction. If X is not a permutator, by the induction hypothesis, the number of permutators in M is not greater than $(3\mathit{asize}(M_1) - 2)/4 + \cdots + (3\mathit{asize}(M_k) - 2)/4 < (3\mathit{asize}(M) - 2)/4$. If X is a permutator, then by the assumption on the form of M, $k \geq 1$ and M_1 is of the form $Y\, N_1 \ldots, N_\ell$ where either Y is not a permutator, or Y is a permutator and $\ell \geq 2$. In the former case, the number of permutators in M is bounded by:

$$\begin{array}{l} 1 + (\sum_{i=1}^{\ell}(3\mathit{asize}(N_i) - 2)/4) + (\sum_{j=2}^{k}(3\mathit{asize}(M_j) - 2)/4) \\ = (3(2 + \sum_{i=1}^{\ell} \mathit{asize}(N_i) + \sum_{j=2}^{k} \mathit{asize}(M_j)) - 2 - 2\ell - 2(k-1))/4 \\ \leq (3\mathit{asize}(M) - 2)/4 \end{array}$$

In the latter case, the number of permutators in M is bounded by:

$$\begin{array}{l} 2 + (\sum_{i=1}^{\ell}(3\mathit{asize}(N_i) - 2)/4) + (\sum_{j=2}^{k}(3\mathit{asize}(M_j) - 2)/4) \\ = (3(2 + \sum_{i=1}^{\ell} \mathit{asize}(N_i) + \sum_{j=2}^{k} \mathit{asize}(M_j)) + 2 - 2\ell - 2(k-1))/4 \\ = (3\mathit{asize}(M) + 2 - 2\ell - 2(k-1))/4 \\ \leq (3\mathit{asize}(M) - 2)/4 \qquad \text{(by } k \geq 1, \ell \geq 2\text{)} \end{array}$$

□

Theorem 7. *Let M be an applicative combination of pure linear λ-terms (that may include extended permutators) each of which is in $\beta\eta$ normal form. Suppose that $\mathcal{K} \vdash_{\mathtt{L}} M : \mathtt{o}$ with $codom(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$, and that M contains neither (i) a consecutive application of permutators (i.e. a subterm of the form $L_1(L_2\,N)$ where L_1 and L_2 are permutators) nor (ii) a permutator in an argument position (i.e., a subterm of the form $N\,L$ where L is a permutator). Then $asize(M) \leq 28\mathbf{w}(\mathcal{K})$.*

Proof. By Lemma 31, the number of permutators in M is bounded by $(3asize(M)-2)/4$. Let M' be the term obtained by (recursively) replacing every term of the form $L\,N$ (where L is a permutator) with N. Since L has a type of the form $\kappa \to \kappa$, M' is also a well-typed linear term, and M' does not contain permutators. Thus by Theorem 6, $asize(M') \leq 7\mathbf{w}(\mathcal{K}) - 6$. By Lemma 31, we have $asize(M') \geq asize(M) - (3asize(M) - 2)/4 = (asize(M) + 2)/4$. Thus, we have $(asize(M) + 2)/4 \leq 7\mathbf{w}(\mathcal{K}) - 6$, which implies $asize(M) \leq 28\mathbf{w}(\mathcal{K})$. □

Theorem 8. *Let M be an applicative combination of pure linear λ-terms (that may include permutators) each of which is in $\beta\eta$ normal form. Suppose that $\mathcal{K} \vdash_{\mathtt{L}} M : \mathtt{o}$ with $codom(\mathcal{K}) \subseteq \{\mathtt{o}, \mathtt{o} \to \mathtt{o} \to \mathtt{o}\}$, and that M contains neither (i) a consecutive application of permutators (i.e. a subterm of the form $L_1(L_2\,N)$ where L_1 and L_2 are permutators) nor (ii) a permutator in an argument position (i.e., a subterm of the form $N\,L$ where L is a permutator). If $M \longrightarrow^* \pi \not\longrightarrow$ (hence π is a normal form that corresponds to a tree), then $asize(M) \leq 28|\pi|$.*

Proof. By subject reduction, $\mathcal{K} \vdash_{\mathtt{L}} \pi : \mathtt{o}$, which implies $|\pi| \geq \mathbf{w}(\mathcal{K})$. Thus, by Theorem 7, we obtain $asize(M) \leq 28\mathbf{w}(\mathcal{K}) \leq 28|\pi|$ as required. □

We can now prove Lemma 4.

Proof of Lemma 4. Let $c_2 = 28$. Suppose $p{\downarrow_{\lambda,\mathtt{lin}}}$ and $p(\longrightarrow_{\mathcal{G},\mathtt{lin}} \cup \longrightarrow_{\lambda,\mathtt{lin}})^*\pi$. By Lemma 6, there exists a pure λ-term M and a type environment Δ that satisfies the conditions of Lemma 6, in particular, $|p| \leq asize(M)$ holds. By Theorem 8, we have $|p| \leq asize(M) \leq 28|\pi|$ as required.

C.5 Proof of Theorem 2

We can now prove Theorem 2. Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be an order-2 extended grammar over $\mathcal{C}$ and $\vdash_{\mathcal{C}} \mathcal{G} \Rightarrow \mathcal{G}'$. Let c be $7 \times 28 = 196$. Suppose $\pi \in \mathcal{L}(\mathcal{G})$. By Theorem 1, there exists a reduction sequence $S^g_{\mathtt{o}} \Longrightarrow_{\mathcal{G}',\mathtt{lin}} p_1 \Longrightarrow_{\mathcal{G}',\mathtt{lin}} \cdots \Longrightarrow_{\mathcal{G}',\mathtt{lin}} p_n \Longrightarrow_{\mathcal{G}',\mathtt{lin}} \pi$ where no intermediate term contains more than two consecutive applications of combinators. Thus, by Lemma 1, there also exists a corresponding reduction sequence $S_{\mathtt{o}} = \overline{p_0} \Longrightarrow_{\mathcal{G}} \overline{p_1} \Longrightarrow_{\mathcal{G}} \cdots \Longrightarrow_{\mathcal{G}} \overline{p_n} = \pi$. Since $|\overline{p}| \leq |p|$, it remains to show $|p| \leq c|\pi|$ holds for every intermediate term p. This follows from Lemmas 3 and 4. □