On Average-Case Hardness of Higher-Order Model Checking

Yoshiki Nakamura
Tokyo Institute of Technology, Japan
nakamura.yoshiki.ny@gmail.com

Kazuyuki Asada
Tohoku University, Sendai, Japan
asada@riec.tohoku.ac.jp

Naoki Kobayashi
The University of Tokyo, Japan
koba@is.s.u-tokyo.ac.jp

Ryoma Sin'ya
Akita University, Japan
ryoma@math.akita-u.ac.jp

Takeshi Tsukada
The University of Tokyo, Japan
tsukada@is.s.u-tokyo.ac.jp

Abstract

We study a mixture between the *average* case and worst case complexities of higher-order model checking, the problem of deciding whether the tree generated by a given λY-term (or equivalently, a higher-order recursion scheme) satisfies the property expressed by a given tree automaton. Higher-order model checking has recently been studied extensively in the context of higher-order program verification. Although the worst-case complexity of the problem is k-EXPTIME complete for order-k terms, various higher-order model checkers have been developed that run efficiently for typical inputs, and program verification tools have been constructed on top of them. One may, therefore, hope that higher-order model checking can be solved efficiently in the *average* case, despite the worst-case complexity. We provide a negative result, by showing that, under certain assumptions, for almost every term, the higher-order model checking problem specialized for the term is k-EXPTIME hard with respect to the size of automata. The proof is based on a novel intersection type system that characterizes terms that do not contain any useless subterms.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Higher-order model checking, average-case complexity, intersection type system

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.21

Related Version A full version of the paper is available at `https://www.kb.is.s.u-tokyo.ac.jp/~koba/papers/OnAverageCaseHOMC.pdf` [20].

Funding This work was supported by JSPS Kakenhi 15H05706 and JSPS Kakenhi 20H00577.

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

1 Introduction

Higher-order model checking [12, 21, 24] asks whether the (possibly infinite) tree generated by a given λY-term (or equivalently, a higher-order recursion scheme) is accepted by a given tree automaton. The problem was shown to be decidable by Ong in 2006 [21], and has been applied to higher-order program verification [15, 16, 22, 19]. Although the worst-case complexity of

higher-order model checking is k-EXPTIME complete (where k is the type-theoretic order of the given λY-term), practical higher-order model checkers have been developed that run fast for many typical inputs. They lead to the development of various automated verification tools for higher-order functional programs.

In view of the situation above, we are interested in the following question: why do higher-order model checkers run efficiently, despite the extremely high worst case complexity? There are a couple of known reasons. First, the worst-case time complexity of higher-order model checking is actually polynomial in the size of a given term, provided that the other parameters (the largest order and arity of functions, and the size of an automaton) are fixed [17]. Second, linear functions do not blow up the complexity [5]. These reasons alone, however, do not fully explain why higher-order model checking works in practice. For example, for the first point above, the constant factor determined by the other parameters is huge.

In the present paper, we consider another possibility: higher-order model checking may actually be easy in the *average* case; in other words, it may be the case that hard instances that cost k-EXPTIME are sparse and many of the instances of higher-order model checking can be solved more efficiently. We give a somewhat negative result on that possibility. For each term t of the λY-calculus, we consider the following higher-order model checking problem specialized to t:

$\mathrm{HOMC}(t, \cdot)$: Given a tree automaton $\mathcal{A}$, decide whether the tree generated by t is accepted by $\mathcal{A}$.

Our main result is that for *almost every* term t of order-k that is sufficiently large, $\mathrm{HOMC}(t, \cdot)$ is k-EXPTIME hard. A little more precisely, we prove that, for the set $\mathtt{Terms}_{n,k}$ of terms of size n and order k (modulo certain additional conditions that we explain later), the ratio of "hard" terms:

$$\frac{\#\{t \in \mathtt{Terms}_{n,k} \mid \mathrm{HOMC}(t, \cdot) \text{ is } k\text{-EXPTIME hard}\}}{\#\mathtt{Terms}_{n,k}}$$

tends to 1 if $n \to \infty$ (where $\#S$ denotes the cardinality of a set S). In other words, if we pick up a term randomly according to the uniform distribution over $\mathtt{Terms}_{n,k}$, it is likely that there exists a bad automaton $\mathcal{A}$ such that $\mathrm{HOMC}(t, \mathcal{A})$ is very hard. Note that this is a mixture between the average case and worst-case analysis: the result above says that in the *average case* on the choice of a term t, the complexity of $\mathrm{HOMC}(t, \cdot)$ is k-EXPTIME hard in the *worst-case* on the choice of an automaton.

In order to make the above analysis meaningful, we have to carefully define the set $\mathtt{Terms}_{n,k}$ of terms. To see why, consider a term of the form $(\lambda x.\mathtt{c})t$, where $\mathtt{c}$ is a nullary tree constructor. The term generates the singleton tree $\mathtt{c}$; so, no matter how large t is, the problem $\mathrm{HOMC}((\lambda x.\mathtt{c})t, \cdot)$ is easy. Thus, if we include such terms in $\mathtt{Terms}_{n,k}$, the ratio of hard instances above would not be 1 for the trivial reason. In the context of applications of higher-order model checking to program verification, however, such instances are unlikely to appear: a λY-term corresponds to a program, and it is unlikely that one writes a program that contains such a huge useless term t. (It might be the case for machine-generated programs, but even in that case, one can apply simple preprocessing to remove such useless terms before invoking a costly higher-order model checking algorithm.) We, therefore, exclude out, from $\mathtt{Terms}_{n,k}$, terms that contain any useless subterms. Here, a subterm t_1 of t is useless if replacing t_1 with another term never changes the tree generated by t. (We will impose further conditions such as the number of variables, which will be explained in Section 2.)

Once the set $\mathtt{Terms}_{n,k}$ is properly chosen as explained above, our main result can be proved as follows. First, according to Kobayashi and Ong's work on the complexity of higher-order model checking [18], there exists an order-k "hard" term $t_{\mathrm{HARD},k}$ such that

HOMC($t_{\text{HARD},k}, \cdot$) is k-EXPTIME complete. Second, according to Asada et al.'s work on quantitative analysis on λ-terms [1], any sufficiently large term t can be decomposed into the form $E[C_1, \ldots, C_m]$ for sufficiently many contexts $C_1, \ldots, C_m$, where each C_i is large enough to be replaced by a context, say C'_i, that contains the hard term $t_{\text{HARD},k}$, without changing the term size. Thus, by using their argument (which originates from the so called "infinite monkey theorem" stating that almost every word contains any given word), we can deduce that almost every sufficiently large term contains the hard term $t_{\text{HARD},k}$, *if we ignore the condition that useless terms should be excluded*. Finally (and most importantly), we can choose a context C'_i that contains the hard term, so that if $E[C_1, \ldots, C_i, \ldots, C_m]$ belongs to $\texttt{Terms}_{n,k}$ (and therefore does not contain any useless subterms), then so does $E[C_1, \ldots, C'_i, \ldots, C_m]$.

To obtain the last part of the result, we develop a novel intersection type system that completely characterizes the set of terms that do not contain useless terms, in the sense that a closed term t is typable if and only if t does not contain any useless term. This type system is one of the main contributions of the present paper, and may be of independent interest. Type systems for useless code elimination have been studied before [6, 7, 13] (in particular, Damiani [7] used intersection types), but the complete characterization was not known, to our knowledge.

The rest of this paper is structured as follows. Section 2 provides formal definitions of λY-terms and the higher-oder model checking. Section 3 states our main result and gives a proof outline. Sections 4–6 prove the theorem. Section 7 discusses related work, and Section 8 concludes this article.

2 Preliminaries

For a map f, we write $\mathrm{dom}(f)$ for the domain of f and $\mathrm{rng}(f)$ for the range of f. We denote by $\mathbb{N}$ the set of non-negative integers and by $\mathbb{N}_+$ the set of positive integers. For $m, n \in \mathbb{N}$, we write $[m, n]$ for the set $\{i \in \mathbb{N} \mid m \leq i \leq n\}$, and $[n]$ for $[1, n]$; note that $[0] = \emptyset$. The cardinality of a set A is denoted by $\#(A)$. We use $A \uplus B$ instead of $A \cup B$ if sets A and B are disjoint. For a set A, we write A^* for the set of finite sequences consisting of elements of A. An *L-labeled tree* is a partial map T from $\mathbb{N}_+^*$ to L such that, for every $\langle \alpha, i \rangle \in \mathbb{N}_+^* \times \mathbb{N}_+$, if $\alpha \cdot i \in \mathrm{dom}(T)$, then $\{\alpha, \alpha \cdot 1, \ldots, \alpha \cdot (i-1)\} \subseteq \mathrm{dom}(T)$. An L-labeled tree T is called *finite* if $\mathrm{dom}(T)$ is finite. We write $\mathtt{r}_T(\alpha)$ for the number of children of a node α in T, i.e., $\mathtt{r}_T(\alpha) = \#\{i \in \mathbb{N}_+ \mid \alpha \cdot i \in \mathrm{dom}(T)\}$. A *ranked alphabet* Σ is a map from a finite set of symbols to $\mathbb{N}$. We call $\Sigma(a)$ the *rank* of a. A $\mathrm{dom}(\Sigma)$-labeled tree T is called a *Σ-ranked tree* (*Σ-tree*, for short) if, for every $\alpha \in \mathrm{dom}(T)$, $\mathtt{r}_T(\alpha) = \Sigma(T(\alpha))$.

2.1 λY-Terms as Tree Generators

In this subsection, we introduce (simply-typed) λY-terms [27] as generators of (possibly infinite) Σ-trees. In the context of higher-order model checking, higher-order recursion schemes have originally been used as generators of trees [12, 21], but the λY-terms (with constants of order up to 1 as tree constructors), which are equi-expressive with higher-order recursion schemes, (see, e.g., [25]), have also been used in later studies on higher-order model checking [24]. For the purpose of the present paper, we find it more convenient to use λY-terms.

Let Σ be a ranked alphabet. Each $a \in \mathrm{dom}(\Sigma)$ is called a *tree constructor*. We use meta-variables a, b, c for tree constructors (and $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$ for concrete symbols). The set of *simple types* is defined by: $\kappa ::= \mathtt{o} \mid \kappa_1 \to \kappa_2$. The *ground type* $\mathtt{o}$ is the type of

trees. The *order* and *arity* of a simple type κ, written $\mathtt{ord}(\kappa)$ and $\mathtt{ar}(\kappa)$ respectively, are defined by: $\mathtt{ord}(\kappa_1 \to \cdots \to \kappa_n \to \mathtt{o}) \triangleq \max(\{0\} \cup \{\mathtt{ord}(\kappa_i) + 1 \mid 1 \leq i \leq n\})$ and $\mathtt{ar}(\kappa_1 \to \cdots \to \kappa_n \to \mathtt{o}) \triangleq n$, where $n \geq 0$. Let $\mathcal{V}$ be a countably infinite set, which is ranged over by x, y, z.

▶ **Definition 1** (λY-terms). *The set of* (λY-)terms *(over Σ) is defined by:*

$$t ::= x^\kappa \mid \lambda x^\kappa.t \mid \lambda\_^\kappa.t \mid t_1\, t_2 \mid \mathbf{Y}^\kappa t \mid a(t_1, \ldots, t_{\Sigma(a)}) \mid \bot^\kappa.$$

We call elements of $\mathcal{V} \uplus \{\_\}$ *variables* and use meta-variables $\bar{x}, \bar{y}, \bar{z}$ for them. As in the standard λY-calculus, the constructor $\mathbf{Y}^\kappa$ may be considered a fixpoint operator of type $(\kappa \to \kappa) \to \kappa$. The special variable '_' denotes an unused variable (hence can occur only in a binder, not in the body of a function). For each type κ, we have a special term $\bot^\kappa$, which intuitively represents an unused term and will play an important role in the definition of minimal terms. We often omit type annotations (for example, $\lambda x^\kappa.x^\kappa$ is just written $\lambda x.x$). For a term t, we write $\mathbf{FV}(t)$ for the set of all the free variables of t.

A *simple type environment* Γ is a finite partial map from $\mathcal{V}$ (recall that the special variable _ does not belong to $\mathcal{V}$) to the set of simple types. We simply write $\Gamma, x : \kappa$ for $\Gamma \uplus \{x \mapsto \kappa\}$. The type judgment relation $\Gamma \vdash_{\mathrm{ST}} t : \kappa$ is inductively defined by the following rules:

$$\frac{}{x : \kappa \vdash_{\mathrm{ST}} x^\kappa : \kappa}(\mathrm{Var}) \quad \frac{\Gamma, x : \kappa \vdash_{\mathrm{ST}} t : \kappa'}{\Gamma \vdash_{\mathrm{ST}} \lambda x^\kappa.t : \kappa \to \kappa'}(\mathrm{Abs1}) \quad \frac{\Gamma \vdash_{\mathrm{ST}} t : \kappa'}{\Gamma \vdash_{\mathrm{ST}} \lambda \bar{x}^\kappa.t : \kappa \to \kappa'}(\mathrm{Abs2}) \quad \frac{}{\emptyset \vdash_{\mathrm{ST}} \bot^\kappa : \kappa}(\bot)$$

$$\frac{\Gamma_1 \vdash_{\mathrm{ST}} t : \kappa \to \kappa' \quad \Gamma_2 \vdash_{\mathrm{ST}} s : \kappa}{\Gamma_1 \cup \Gamma_2 \vdash_{\mathrm{ST}} t\, s : \kappa'}(\mathrm{App}) \quad \frac{\Gamma_1 \vdash_{\mathrm{ST}} t_1 : \mathtt{o} \ \ldots\ \Gamma_n \vdash_{\mathrm{ST}} t_n : \mathtt{o}}{\bigcup_{i \in [n]} \Gamma_i \vdash_{\mathrm{ST}} a(t_1, \ldots, t_n) : \mathtt{o}}(a) \quad \frac{\Gamma \vdash_{\mathrm{ST}} t : \kappa \to \kappa}{\Gamma \vdash_{\mathrm{ST}} \mathbf{Y}^\kappa t : \kappa}(\mathbf{Y})$$

Henceforth, we only consider *well-typed* terms (i.e., terms t such that $\Gamma \vdash_{\mathrm{ST}} t : \kappa$ for some $\langle \Gamma, \kappa \rangle$). Note that for every well-typed term t, there is a unique pair $\langle \Gamma, \kappa \rangle$ such that $\Gamma \vdash_{\mathrm{ST}} t : \kappa$; and moreover, its derivation tree is also uniquely determined. We sometimes annotate a term with its type, like t^κ, when t has type κ (under a certain type environment). We say that t is *closed* if $\Gamma = \emptyset$; and that t is *ground-typed* if $\kappa = \mathtt{o}$.

▶ **Definition 2.** *The* (call-by-name) reduction relation $\longrightarrow$ *is defined as the least binary relation on well-typed terms (up to α-equivalence) closed under the following rules, where we write $t\{s/x\}$ for the term obtained from t by substituting s for all the free occurrences of x in a capture-avoiding manner:*

(β) $(\lambda \bar{x}.t)\, s \longrightarrow t\{s/\bar{x}\}$; $(\mathbf{Y})$ $\mathbf{Y}t \longrightarrow t\,(\mathbf{Y}t)$; $(\bot)$ $\bot^{\kappa_1 \to \kappa_2} t \longrightarrow \bot^{\kappa_2}$;
(App) $tu \longrightarrow t'u$ *if* $t \longrightarrow t'$; (a) $a(t_1, \ldots, t_n) \longrightarrow a(t_1, \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_n)$ *if* $t_i \longrightarrow t_i'$.

We write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$.

The *tree generated by a closed and ground λY-term t* is the one obtained from t by (possibly) infinite rewriting with respect to the above reduction relation. The precise definition is given below.

We write $\Sigma^\bot$ for the ranked alphabet $\Sigma \uplus \{\bot \mapsto 0\}$. We define *the binary relation $\sqsubseteq$ on $\Sigma^\bot$-trees* by: $T_1 \sqsubseteq T_2$ if and only if (i) $\mathrm{dom}(T_1) \subseteq \mathrm{dom}(T_2)$ and (ii) for every $\alpha \in \mathrm{dom}(T_1)$, $T_1(\alpha) = \bot$ or $T_1(\alpha) = T_2(\alpha)$. We write $T_1 \sqsubset T_2$ if $T_1 \sqsubseteq T_2$ and $T_1 \neq T_2$. We denote the join of $\{T_i\}_{i \in I}$ with respect to $\sqsubseteq$ by $\bigsqcup_{i \in I} T_i$ if defined.

A term consisting of only tree constructors and $\bot^\circ$ can naturally be regarded as a $\Sigma^\bot$-tree. For example, $\mathtt{b}(\mathtt{c}, \mathtt{a}(\bot^\circ))$ can be regarded as the $\Sigma^\bot$-tree: $\{\epsilon \mapsto \mathtt{b}, 1 \mapsto \mathtt{c}, 2 \mapsto \mathtt{a}, 2 \cdot 1 \mapsto \bot\}$; hence we identify finite trees and terms consisting of tree constructors and $\bot^\circ$ below. For

each closed and ground-typed term t, the $\Sigma^{\perp}$-tree $t^{\perp}$ is defined by: $t^{\perp} \triangleq a(t_1^{\perp}, \ldots, t_{\Sigma(a)}^{\perp})$ if $t = a(t_1, \ldots, t_{\Sigma(a)})$; and $t^{\perp} \triangleq \perp$ otherwise. The *value tree* of a closed and ground-typed term t, written $T(t)$, is defined by: $T(t) \triangleq \bigsqcup\{s^{\perp} \mid t \longrightarrow^* s\}$. For example, consider the value tree of $(\mathbf{Y}t_1)\mathtt{c}$ where $t_1 = \lambda f^{\mathtt{o}\to\mathtt{o}}.\lambda x^{\mathtt{o}}.\mathtt{b}(x, f(\mathtt{a}(x)))$. By applying the reduction rules $(\mathbf{Y})$ and (β), we can obtain the following reduction sequence

$$(\mathbf{Y}t_1)\mathtt{c} \longrightarrow t_1(\mathbf{Y}t_1)\mathtt{c} \longrightarrow^* \mathtt{b}(\mathtt{c}, (\mathbf{Y}t_1)(\mathtt{a}(\mathtt{c}))) \longrightarrow^* \mathtt{b}(\mathtt{c}, \mathtt{b}(\mathtt{a}(\mathtt{c}), (\mathbf{Y}t_1)(\mathtt{a}(\mathtt{a}(\mathtt{c})))))$$

and observe that $T(t)$ is the infinite tree of the form $\mathtt{b}(\mathtt{c}, \mathtt{b}(\mathtt{a}(\mathtt{c}), \mathtt{b}(\mathtt{a}(\mathtt{a}(\mathtt{c})), \mathtt{b}(\cdots)))))$.

We also define the size and order of a term, which will be used in the complexity analysis.

▶ **Definition 3** (size, order). *The* size *of a term t is defined by: $|x| = |\perp| \triangleq 1$, $|\lambda\tilde{x}.t| = |\mathbf{Y}t| \triangleq 1 + |t|$, $|t_1\, t_2| \triangleq 1 + |t_1| + |t_2|$, and $|a(t_1, \ldots, t_{\Sigma(a)})| \triangleq 1 + \sum_{i\in[\Sigma(a)]} |t_i|$. The* order *of a term t, written $\mathtt{ord}\,(t)$, is defined by:*

$$\mathtt{ord}\,(t) \triangleq \max(\{0\} \cup \{\mathtt{ord}\,(\kappa) \mid \lambda x^{\kappa}.s \text{ or } \mathbf{Y}^{\kappa}s \text{ is a subterm of } t\}).$$

Note that the size of a variable is a constant; this is appropriate in our context, as we fix the number of variables in the main theorem (Theorem 7).

▶ Remark 4. Our definition of the order of a λY-term given above deviates from the standard definition of the order of a λY-term (where the order of a term is defined as the largest order of the types of subterms) [25]. For example, the order of $Y^{\mathtt{o}}\lambda x^{\mathtt{o}}.\mathtt{a}(x)$ (which generates a unary infinite tree consisting of only $\mathtt{a}$) is 0 in our definition, but it is 1 in the standard definition, because $\lambda x^{\mathtt{o}}.\mathtt{a}(x)$ has type $\mathtt{o} \to \mathtt{o}$, which has order 1. Our definition is motivated to make the order of λY-term equivalent to that of the corresponding higher-order recursion scheme (where the order is defined as the largest order of the types of recursive functions); for example, the above term corresponds to the higher-order recursion scheme consisting of a single rule $S \longrightarrow \mathtt{a}(S)$, whose order is 0. The translation from higher-order recursion schemes to λY-terms given in [25] is order-preserving in our definition, but increases the order by 1 in the definition of [25]. There is also a translation from λY-terms to higher-order recursion schemes that preserves the order in our definition (given an order-k λY-term, reduce all the β-redexes of the form $(\lambda x^{\kappa}.s)t$ with $\mathtt{ord}\,(\kappa) = k$ first, and then apply the translation suggested in [25]; the first phase of β-reductions may incur an exponential blow-up, which can be avoided by appropriately introducing non-terminals to avoid duplications of terms).

2.2 Higher-Order Model Checking

We assume the notion of *alternating parity tree automaton* (*APT* for short): see, e.g., [10]. The precise definition of APT is unnecessary for understanding our technical development in later sections, once you admit the results in this subsection. We recall the definition of higher-order model checking below.

▶ **Definition 5** (higher-order model checking problem). *The* higher-order model checking problem*, written* $\mathrm{HOMC}\,(\cdot,\cdot)$*, is the problem of, given a closed and ground-typed λY-term t over Σ and an APT $\mathcal{A}$ over Σ as input, deciding whether $\mathcal{A}$ accepts $T(t)$. We write* $\mathrm{HOMC}_k(\cdot,\cdot)$ *when the first input is restricted to a term of order-k. We denote by* $\mathrm{HOMC}\,(t,\cdot)$ *the problem obtained by fixing the first input to t, i.e., the problem of, given an APT $\mathcal{A}$ as input, deciding whether $\mathcal{A}$ accepts $T(t)$.*

Ong [21] has shown that the $\mathrm{HOMC}_k(\cdot,\cdot)$ is k-EXPTIME complete (combined complexity) for each $k \geq 0$. The following theorem states the complexity of $\mathrm{HOMC}(t,\cdot)$, which serves as a basis of the present work.

▶ **Theorem 6** ([21] for (1) and [18, Theorem 3.8] for (2)). *For each $k \geq 1$,*
(1) *for every order-k λY-term t, $\mathrm{HOMC}(t,\cdot)$ is decidable in k-EXPTIME; and*
(2) *there exists an order-k λY-term $t_{\mathit{HARD},k}$ such that $\mathrm{HOMC}(t_{\mathit{HARD},k},\cdot)$ is k-EXPTIME hard.*

3 Main Theorem

This section formally states the main result of the paper: for almost every order-k λY-term, the higher-order model checking problem $\mathrm{HOMC}(t,\cdot)$ is k-EXPTIME hard, under a certain assumption, and sketches an overall structure of the proof. We first prepare some auxiliary notations. We denote by $[t]_\alpha$ the α-equivalence class of t. In our quantitative analysis, we count α-equivalent terms at most once (e.g., we do not distinguish $(\lambda x.\lambda y.x)z$ and $(\lambda z.\lambda\_.z)z$). We define $\#\mathtt{vars}(t) \triangleq \min\{\#(\mathbf{V}(t')) \mid t' \in [t]_\alpha\}$, where $\mathbf{V}(t)$ denotes the set of all the variables (except _) occurring in t. Namely, $\#\mathtt{vars}(t)$ is the *minimum number of variables* occurring in the term t, up to α-equivalence. For example, $\#\mathtt{vars}((\lambda x.\lambda y.x)z) = 1$ since the term is α-equivalent to $(\lambda z.\lambda\_.z)z$. The *internal arity* of a term t, written $\mathtt{iar}(t)$, is defined by: $\mathtt{iar}(t) \triangleq \max(\{\mathtt{ar}(\kappa) \mid s^\kappa \text{ is a subterm of } t\})$.

Let $\hat{\Lambda}_n(k,\iota,\xi)$ be the set of all (α-equivalence classes of) closed and ground-typed λY-terms such that[1]

(i) the size is n (i.e., $|t| = n$);
(ii) the order is up to k (i.e., $\mathtt{ord}(t) \leq k$);
(iii) the internal arity is up to ι (i.e., $\mathtt{iar}(t) \leq \iota$);
(iv) the number of variable names is up to ξ (i.e., $\#\mathtt{vars}(t) \leq \xi$); and
(v) the terms are *minimal* (see Section 3.1 below for the definition).

The main theorem is stated as follows.

▶ **Theorem 7** (main theorem). *For each $k \geq 1$, let ι and ξ be sufficiently large natural numbers. Then,*

$$\lim_{n\to\infty} \frac{\#\Big(\{t \in \hat{\Lambda}_n(k,\iota,\xi) \mid \mathrm{HOMC}(t,\cdot) \textit{ is } k\text{-EXPTIME } \textit{hard}\}\Big)}{\#\Big(\hat{\Lambda}_n(k,\iota,\xi)\Big)} = 1.$$

Below we first define the minimality in Section 3.1 and give a proof outline in Section 3.2.

3.1 Minimal Terms

Intuitively, a term is *minimal* if it has no useless subterm. For the formal definition, we first define the relation $\sqsubseteq$ on *terms*, which is analogous to the corresponding relation ($\sqsubseteq$) on trees.

▶ **Definition 8.** *The* approximate relation $\sqsubseteq$ *is the least precongruence (i.e., the relation closed under all the term constructors) such that $\bot^\kappa \sqsubseteq t^\kappa$.*

[1] The set $\hat{\Lambda}_n(k,\iota,\xi)$ implicitly depends on the choice of ranked alphabet Σ. The main theorem holds independently of the choice of Σ unless Σ is unreasonably small.

In other words, $s \sqsubseteq t$ means that s is obtained from t by replacing subterms $t_1^{\kappa_1}, \ldots, t_n^{\kappa_n}$ with $\bot^{\kappa_1}, \ldots, \bot^{\kappa_n}$. We write $s \sqsubset t$ if $s \sqsubseteq t$ and $s \neq t$. We denote the join of $\{t_i\}_{i\in I}$ with respect to $\sqsubseteq$ (i.e., the least upper bound of $\{t_i\}_{i\in I}$ with respect to $\sqsubseteq$) by $\bigsqcup_{i\in I} t_i$ if defined, and we sometimes write $t_1 \sqcup \ldots \sqcup t_n$ for $\bigsqcup_{i\in[n]} t_i$. For example, $(\lambda x.\mathtt{b}(x,\bot)) \sqcup (\lambda x.\mathtt{b}(\bot,x)) = \lambda x.\mathtt{b}(x,x)$. Note that, with respect to $\Sigma^{\bot}$-tree terms, the relation $\sqsubseteq$ on terms is equivalent to the relation $\sqsubseteq$ on $\Sigma^{\bot}$-trees.

▶ **Definition 9.** *A closed and ground-typed term t is* minimal *if for every $s \sqsubset t$, $T(s) \neq T(t)$.*

In other words, a term t is *not minimal* if there exists s obtained by replacing a non-$\bot$ subterm u of t with $\bot$ such that $T(s) = T(t)$.

▶ **Example 10.** Let $t = (\lambda x.\lambda y.x)\ \mathtt{a}\ u$, with $u \neq \bot$. Then the value tree $T(t) = \mathtt{a}$ (since $(\lambda x.\lambda y.x)\ \mathtt{a}\ u \longrightarrow (\lambda y.\mathtt{a})\ u \longrightarrow \mathtt{a}$). Note that the subterm u is "useless"; indeed the term $s = (\lambda x.\lambda y.x)\ \mathtt{a}\ \bot$, obtained from t by replacing u with $\bot$, also generates $\mathtt{a}$. Thus, t is not minimal. In contrast, s is minimal. In fact, any term obtained by replacing a non-$\bot$ subterm of s with $\bot$ (such as $(\lambda x.\lambda y.\bot)\ \mathtt{a}\ \bot$) fails to generate $\mathtt{a}$.

The following proposition gives an important property of minimal terms. We write $t' \preceq t$ when t' is a subterm of a term t.

▶ **Proposition 11.** *Let t be a closed and ground-typed term. If t is minimal, then for every non-$\bot$, closed and ground-typed subterm $s \preceq t$, its value tree $T(s)$ is a subtree of $T(t)$.*

This property is intuitively obvious. Since t is minimal, the subterm s assumed to be non-$\bot$ must be used in the computation of the value tree $T(t)$. As s is closed and ground-typed, the only way to use s is to place its value tree $T(s)$ somewhere in $T(t)$; hence the proposition. For a formal proof, see the full version [20].

3.2 Proof Outline

For each k, let $t_{\mathrm{HARD},k}$ be an order-k closed, ground-typed term such that the problem $\mathrm{HOMC}(t,\cdot)$ is k-EXPTIME hard. The existence of $t_{\mathrm{HARD},k}$ is guaranteed by Theorem 6 (2). We can assume, without loss of generality, that $t_{\mathrm{HARD},k}$ is minimal; otherwise take a minimal element $t'_{\mathrm{HARD},k}$ of $\{s \mid T(s) = T(t_{\mathrm{HARD},k})\}$. Theorem 7 follows immediately from Lemmas 12 and 13 below, which respectively state: (a) for each order k, every order-k minimal term containing the "hard" term $t_{\mathrm{HARD},k}$ as a subterm yields k-EXPTIME-hardness for the higher-order model checking problem; and (b) almost every minimal term of order k contains the "hard" term $t_{\mathrm{HARD},k}$ as a subterm.

▶ **Lemma 12.** *Let $k \geq 1$. For every* minimal *λY-term $t \succeq t_{\mathrm{HARD},k}$, $\mathrm{HOMC}(t,\cdot)$ is k-EXPTIME hard.*

Proof. Assume that $t \succeq t_{\mathrm{HARD},k}$. Then $T(t) \succeq T(t_{\mathrm{HARD},k})$ by Proposition 11, i.e. $T(t_{\mathrm{HARD},k}) = (T(t)\restriction_\alpha)$ for some $\alpha \in \mathrm{dom}(T(t))$ where $(T\restriction_\alpha)$ denotes the subtree of T induced by the node α. Let c be the length of α. For any APT $\mathcal{A}$, we can construct an automaton $\mathcal{A}\restriction_\alpha$ by adding c states to $\mathcal{A}$ and replacing the initial state so that $\mathcal{A}\restriction_\alpha$ accepts T if and only if $\mathcal{A}$ accepts $T\restriction_\alpha$ (intuitively, $\mathcal{A}\restriction_\alpha$ first moves to the node α then behaves like $\mathcal{A}$). Then the polynomial-time function $\mathcal{A} \mapsto (\mathcal{A}\restriction_\alpha)$ gives a polynomial-time reduction from $\mathrm{HOMC}(t_{\mathrm{HARD},k},\cdot)$ to $\mathrm{HOMC}(t,\cdot)$ The lemma follows from k-EXPTIME-hardness of $\mathrm{HOMC}(t_{\mathrm{HARD},k},\cdot)$. ◀

▶ **Lemma 13.** *For each $k \geq 1$, let ι and ξ be sufficiently large natural numbers. Then,*

$$\lim_{n\to\infty} \frac{\#\Big(\{t \in \hat{\Lambda}_n(k,\iota,\xi) \mid t \succeq t_{\mathit{HARD},k}\}\Big)}{\#\Big(\hat{\Lambda}_n(k,\iota,\xi)\Big)} = 1.$$

It remains to show Lemma 13. To this end, we introduce the following lemma (where the precise definition of *second-order contexts* will be given in Section 4).

▶ **Lemma 14.** *For each $k \geq 1$, let ι and ξ be sufficiently large natural numbers. There exists m such that the following holds: Let $n \geq m$, E be any second-order linear context, and C be any affine context such that $|C| \geq m$ and $E[C] \in \hat{\Lambda}_n(k,\iota,\xi)$. Then there exists an affine context $D \succeq t_{\mathit{HARD},k}$ such that $E[D] \in \hat{\Lambda}_n(k,\iota,\xi)$.*

We show how Lemma 13 follows from Lemma 14 in Section 4. We then introduce a new intersection type system that characterizes the minimality in Section 5, and use it to prove Lemma 14 in Section 6.

4 Infinite Monkey Theorem for Minimal Terms

We sketch a proof of Lemma 13 (modulo Lemma 14) in this section; see [20] for the full proof. The proof is analogous to that of the following, so-called *infinite monkey theorem* (a.k.a. "Borges's theorem" [9, p.61, Note I.35]) for words:

▶ **Theorem 15.** *Let Σ be a finite alphabet. For any word $x \in \Sigma^*$, almost all words contain x as a subword, i.e.*

$$\lim_{n\to\infty} \frac{\#(\{w \in \Sigma^n \mid w = uxv \text{ for some } u, v \in \Sigma^*\})}{\#(\Sigma^n)} = 1.$$

The theorem above follows from the following reasoning: Any word w can be decomposed into the form $w_1 w_2 \cdots w_p w'$ where $|w_i| = |x|$ and $|w'| < |x|$. If we pick w randomly, the probability that w_i coincides with x is $(\frac{1}{|\Sigma|})^{|x|}$; hence the probability that w contains x is at least $1 - (1 - (\frac{1}{|\Sigma|})^{|x|})^p$, which tends to 1 when n tends to infinity. For the purpose of proving Lemma 13, we analogously decompose each term t to the form $E[C_1, \ldots, C_p]$ (where E and C_i respectively correspond to w' and w_i above), by using the tree decomposition in [1]. We can then use Lemma 14 to prove Lemma 13. The hardest part is actually to prove Lemma 14, which is deferred to Section 6.

We first need to prepare some definitions. In order to make use of the tree decomposition function Φ_m in [1], below we regard a λY-term over Σ as a $\Sigma_{\hat{\Lambda}(k,\iota,\xi)}$-tree where $\Sigma_{\hat{\Lambda}(k,\iota,\xi)}$ is an extension of Σ defined by:

$$\begin{aligned}\Sigma_{\hat{\Lambda}(k,\iota,\xi)} \triangleq \Sigma &\uplus \{x \mapsto 0 \mid x \in \mathcal{V}_\xi\}\\ &\uplus \{\lambda\bar{x}^\kappa \mapsto 1 \mid \bar{x} \in \mathcal{V}_\xi \uplus \{\_\}, \mathtt{ord}(\kappa) \leq k, \mathtt{iar}(\kappa) \leq \iota\}\\ &\uplus \{@ \mapsto 2\} \uplus \{\mathbf{Y}^\kappa \mapsto 1, \bot^\kappa \mapsto 0 \mid \mathtt{ord}(\kappa) \leq k, \mathtt{iar}(\kappa) \leq \iota\}\end{aligned}$$

Here, $\mathcal{V}_\xi = \{x_1, \cdots, x_\xi\}$ is a finite subset of $\mathcal{V}$ and the symbol @ represents the application operation.

Next, we recall the notion of contexts and second-order contexts used in the decomposition. A *context* is a tree with special leaves $[\,]$ called *holes*. Formally, the set of contexts over Σ is given by

$$C ::= [\,] \mid a(C_1, \ldots, C_{\Sigma(a)}),$$

where a ranges over $\mathrm{dom}(\Sigma)$. We call a context with k holes a *k-context*, and call a context *affine* if it is a 0- or 1-context. The *size* of a context C, denoted by $|C|$, is inductively defined by: $|[\,]| \triangleq 0$ and $|a(C_1, \ldots, C_{\Sigma(a)})| \triangleq 1 + |C_1| + \cdots + |C_{\Sigma(a)}|$. For a k-context C and contexts $\overrightarrow{C} = C_1 \cdots C_k$, we write $C[\overrightarrow{C}]$ or $C[C_1, \ldots, C_k]$ for the context obtained by replacing each occurrence of $[\,]$ in C with C_i in the left-to-right order.

A *second-order context* is an expression having holes of the form $[\![]\!]^n_k$ (called *second-order holes*), which should be filled with a k-context of size n. Formally, the set of second-order contexts over Σ, ranged over by E, is defined by:

$$E ::= [\![]\!]^n_k[E_1, \ldots, E_k] \;\mid\; a(E_1, \ldots, E_{\Sigma(a)}) \quad (a \in \mathrm{dom}(\Sigma)).$$

We write $\mathtt{shn}\,(E)$ for the number of the second-order holes in E, and $E.i$ for the i-th leftmost second-order hole in E.

▶ **Definition 16** (substitution for second-order contexts)**.** *For a context C and a second-order hole $[\![]\!]^n_k$, we write $C : [\![]\!]^n_k$ if C is a k-context of size n. For a second-order context E and a sequence of contexts $\overrightarrow{C} = C_1 \cdots C_{\mathtt{shn}(E)}$ such that $C_i : E.i$ for each $i \in [\mathtt{shn}\,(E)]$, we write $E[\overrightarrow{C}]$ or $E[C_1, \ldots, C_{\mathtt{shn}(E)}]$ for the tree which can be obtained by replacing each occurrence of $[\![]\!]$ in E with C_i in the left-to-right manner (and by interpreting the syntactical bracket $[-]$ as the substitution operation for usual contexts), where $\#\left(\overrightarrow{C_i}\right) = \mathtt{shn}\,(E_i)$ for each i:*

$$([\![]\!]^n_k[E_1, \ldots, E_k])\,[C \cdot \overrightarrow{C_1} \cdots \overrightarrow{C_k}] \triangleq C[E_1[\overrightarrow{C_1}], \ldots, E_k[\overrightarrow{C_k}]]$$
$$(a(E_1, \ldots, E_{\Sigma(a)}))[\overrightarrow{C_1} \cdots \overrightarrow{C}_{\Sigma(a)}] \triangleq a(E_1[\overrightarrow{C_1}], \ldots, E_{\Sigma(a)}[\overrightarrow{C}_{\Sigma(a)}]).$$

We can use the decomposition function Φ_m (where $m > 0$ is an integer parameter) introduced in [1] to uniquely decompose (the tree representation of) a λY-term t to $(E, C_1, \ldots, C_k)$ such that (i) E is a second-order context, (ii) C_i's are affine contexts such that $m \le |C_i| \le rm$ (where r is the largest arity of the symbols in $\Sigma_{\hat{\Lambda}(k,\iota,\xi)}$), (iii) $t = E[C_1, \ldots, C_k]$, (iv) $k \ge \frac{|t|}{2rm}$, (v) $\Phi_m(E[C_1, \ldots, C_{i-1}, D_i, C_{i+1}, \ldots, C_k]) = (E, C_1, \ldots, C_{i-1}, D_i, C_{i+1}, \ldots, C_k)$ for any "good" context D_i (see [1, 20] for the definition of "goodness").

▶ **Example 17.** The term $\mathsf{a}((\lambda\_.\mathsf{a}((\lambda x.\mathsf{a}(x))(\mathbf{Y}\,\lambda y.\mathsf{a}(y))))\bot)$ on the left hand side of Figure 1 can be decomposed into the second-order context $\mathsf{a}([\![]\!]^4_1[([\![]\!]^3_0[])(\mathbf{Y}\,([\![]\!]^3_0[]))])$ and affine contexts, as shown on the right hand side.

We are now ready to provide a proof sketch of Lemma 13. Let us define $S^{n,E}_i$ and $\mathcal{E}^n_m$ by:

$$S^{n,E}_i \triangleq \{t \in \hat{\Lambda}_n(k, \iota, \xi) \mid \Phi_m(t) = (E, C_1, \ldots, C_\ell),\ \text{and } t_{\mathrm{HARD},k} \not\preceq C_j \text{ for each } j \in [i]\}$$
$$\mathcal{E}^n_m \triangleq \left\{E \mid \Phi_m(t) = (E, \cdots) \text{ for some } t \in \hat{\Lambda}_n(k, \iota, \xi)\right\}.$$

Below we write $E \star (C_1, \ldots, C_\ell) \in S$ to mean $E[C_1, \ldots, C_\ell] \in S$ and $\Phi_m(E[C_1, \ldots, C_\ell]) = (E, C_1, \ldots, C_\ell)$. If n and m (with $n > m$) are sufficiently large, we can estimate the ratio $\frac{\#\left(S^{n,E}_i\right)}{\#\left(S^{n,E}_{i-1}\right)}$ for $i \in [\mathtt{shn}\,(E)]$ by:

$$\frac{\#\left(S^{n,E}_i\right)}{\#\left(S^{n,E}_{i-1}\right)} = \frac{\sum_{C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}} \#\left(\left\{C_i \mid E \star (C_1, \ldots, C_{\mathtt{shn}(E)}) \in S^{n,E}_{i-1}, t_{\mathrm{HARD},k} \not\preceq C_i\right\}\right)}{\sum_{C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}} \#\left(C_i \mid E \star (C_1, \ldots, C_{\mathtt{shn}(E)}) \in S^{n,E}_{i-1}\right)}$$
$$\le \max_{C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}} \frac{\#\left(\left\{C_i \mid E \star (C_1, \ldots, C_{\mathtt{shn}(E)}) \in S^{n,E}_{i-1}, t_{\mathrm{HARD},k} \not\preceq C_i\right\}\right)}{\#\left(\left\{C_i \mid E \star (C_1, \ldots, C_{\mathtt{shn}(E)}) \in S^{n,E}_{i-1}\right\}\right)}$$
$$\left(\text{by } \frac{\sum_j s_j}{\sum_j r_j} \le \max_j \frac{s_j}{r_j} \text{ if } r_j > 0, s_j \ge 0 \text{ for every } j\right)$$

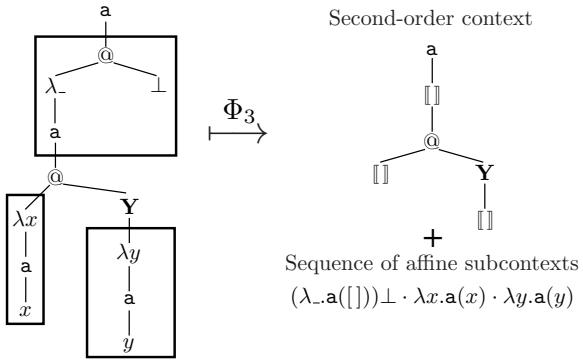

Figure 1 An example of term decomposition. The parts surrounded by rectangles on the left hand side show the extracted affine subcontexts, and the remaining part of the tree is the second-order tree context.

$$
\begin{aligned}
&\leq \max_{C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}} \frac{\#\left(\left\{C_i \mid E \star (C_1,\ldots,C_{\mathtt{shn}(E)}) \in S_{i-1}^{n,E}\right\}\right) - 1}{\#\left(\left\{C_i \mid E \star (C_1,\ldots,C_{\mathtt{shn}(E)}) \in S_{i-1}^{n,E}\right\}\right)} \\
&\qquad \text{(by Lemma 14)} \\
&= \max_{C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}} 1 - \frac{1}{\#\left(\left\{C_i \mid E \star (C_1,\ldots,C_{\mathtt{shn}(E)}) \in S_{i-1}^{n,E}\right\}\right)} \\
&\leq 1 - \frac{1}{\gamma^{rm}}
\end{aligned}
$$

for some $\gamma > 1$. Here, $C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_{\mathtt{shn}(E)}$ in the subscript of max range over the set of contexts for which the denominator $\#\left(\left\{C_i \mid E \star (C_1,\ldots,C_{\mathtt{shn}(E)}) \in S_i^{n,E}\right\}\right)$ is non-zero. The last inequality follows from Property (ii) of Φ_m (that the size of C_i is bounded by rm), and the fact that the number of contexts of a given size ℓ can be bounded by γ^ℓ for some γ.

Thus, we have:

$$
\begin{aligned}
\frac{\#\left(\{t \in \hat{\Lambda}_n(k,\iota,\xi) \mid t_{\mathrm{HARD},k} \not\preceq t\}\right)}{\#\left(\hat{\Lambda}_n(k,\iota,\xi)\right)} &\leq \frac{\sum_{E\in\mathcal{E}_m^n} \#\left(S_{\mathtt{shn}(E)}^{n,E}\right)}{\sum_{E\in\mathcal{E}_m^n} \#\left(S_0^{n,E}\right)} \quad \left(\text{by the properties of } \Phi_m\right) \\
&\leq \max_{E\in\mathcal{E}_m^n} \frac{\#\left(S_{\mathtt{shn}(E)}^{n,E}\right)}{\#\left(S_0^{n,E}\right)} \quad \left(\text{by } \frac{\sum_i s_i}{\sum_i r_i} \leq \max_i \frac{s_i}{r_i}\right) \\
&= \max_{E\in\mathcal{E}_m^n} \frac{\#\left(S_1^{n,E}\right)}{\#\left(S_0^{n,E}\right)} \cdot \frac{\#\left(S_2^{n,E}\right)}{\#\left(S_1^{n,E}\right)} \cdot \cdots\cdots \frac{\#\left(S_{\mathtt{shn}(E)}^{n,E}\right)}{\#\left(S_{\mathtt{shn}(E)-1}^{n,E}\right)}. \\
&\leq \max_{E\in\mathcal{E}_m^n} \left(1 - \frac{1}{\gamma^{rm}}\right)^{\mathtt{shn}(E)} \\
&\leq \left(1 - \frac{1}{\gamma^{rm}}\right)^{\frac{n}{2rm}} \quad \left(\text{by Property (iv) of } \Phi_m\right) \\
&\to 0 \ (\text{as } n \to \infty)
\end{aligned}
$$

Therefore, we obtain

$$\lim_{n\to\infty} \frac{\#\Big(\{t \in \hat{\Lambda}_n(k,\iota,\xi) \mid t \succeq t_{\mathrm{HARD},k}\}\Big)}{\#\Big(\hat{\Lambda}_n(k,\iota,\xi)\Big)} = 1$$

as required.

5 Intersection Types for Minimal Terms

This section introduces an intersection type system for characterizing minimal terms, so that a closed, ground-typed term is typable just if it is minimal. For the terms in Example 10, $(\lambda x.\lambda y.x)\ \mathtt{a}\ \bot$ is typable in the intersection type system but $(\lambda x.\lambda y.x)\ \mathtt{a}\ \mathtt{a}$ is not. This intersection type system will serve as a key tool for proving Lemma 14 in Section 6.

The sets of *prime intersection types* and *intersection types* are defined by:

$$\tau,\sigma \text{ (prime intersection types)} ::= \mathtt{o} \mid \theta \to \tau \qquad \theta,\delta \text{ (intersection types)} ::= \textstyle\bigwedge^{\kappa}\{\tau_1,\ldots,\tau_n\}.$$

Here, $n \geq 0$: Intuitively, $\mathtt{o}$ is the type of terms that generate non-$\bot$ trees that will occur in the value tree. The intersection type $\bigwedge^{\kappa}\{\tau_1,\ldots,\tau_n\}$ describes terms that are used according to types $\tau_1,\ldots,\tau_n$. In particular, $\bigwedge^{\kappa}\emptyset$ is the type of terms that are not used. Note that $\{\tau_1,\ldots,\tau_n\}$ in $\bigwedge^{\kappa}\{\tau_1,\ldots,\tau_n\}$ is a *set* rather than a multiset; thus we consider here idempotent intersection types. The superscript κ (which ranges over the set of simple types) is used for distinguishing between, for example, $\emptyset^{\mathtt{o}}$ and $\emptyset^{\mathtt{o}\to\mathtt{o}}$; we, however, often omit the superscript.

We often write $\bigwedge^{\kappa}_{i\in[n]}\tau_i$ or $\tau_1 \wedge \cdots \wedge \tau_n$ for $\bigwedge^{\kappa}\{\tau_1,\ldots,\tau_n\}$, and $\top^{\kappa}$ (or just $\top$) for $\bigwedge^{\kappa}\emptyset$. For each intersection types $\theta = \bigwedge^{\kappa} S$ and $\delta = \bigwedge^{\kappa} T$, We denote by $\theta \wedge \delta$ the intersection type $\bigwedge^{\kappa}(S \cup T)$. We use $\bar{\theta}$, $\bar{\delta}$ to denote a prime intersection type or an intersection type. An *intersection type environment*, written as Θ or Δ, is a finite partial mapping from $\mathcal{V}$ to the set of intersection types. For each Θ, $x \in \mathcal{V} \setminus \mathrm{dom}(\Theta)$, and θ, we write $(\Theta, x:\theta)$ for $\Theta \cup \{x \mapsto \theta\}$. The *refinement relation* $\bar{\theta} :: \kappa$ (resp. $\Theta :: \Gamma$) is the least relation closed under the following rules, where $n \geq 0$:

$$\frac{}{\mathtt{o} :: \mathtt{o}} \qquad \frac{\tau_1 :: \kappa \quad \ldots \quad \tau_n :: \kappa}{\bigwedge^{\kappa}_{i\in[n]}\tau_i :: \kappa} \qquad \frac{\theta :: \kappa \quad \tau :: \kappa'}{(\theta \to \tau) :: (\kappa \to \kappa')} \qquad \frac{}{\emptyset :: \emptyset} \qquad \frac{\Theta :: \Gamma \quad \theta :: \kappa}{(\Theta, x:\theta) :: (\Gamma, x:\kappa)}.$$

Note that, for each $\bar{\theta}$ (and similarly for Θ), there exists at most one simple type κ such that $\bar{\theta} :: \kappa$. Henceforth we only consider intersection types occurring in this refinement relation (so, we always make the assumption that for each $\bar{\theta}$, $\bar{\theta} :: \kappa$ holds for some κ).

We write $\Theta \wedge \Delta$ for the intersection type environment $\{x \mapsto \Theta(x) \wedge \Delta(x) \mid x \in \mathrm{dom}(\Theta) \cup \mathrm{dom}(\Delta)\}$, where $\Theta(x) = \top^{\kappa}$ (where κ is uniquely determined by $\Delta(x)$) if $x \notin \mathrm{dom}(\Theta)$, and similarly for the case $x \notin \mathrm{dom}(\Delta)$.

The *intersection type judgment relation* $\Theta \vdash t : \bar{\theta}$ is inductively defined by the typing rules in Figure 2. We implicitly assume that, whenever $\Theta \vdash t : \bar{\theta}$ occurs in a rule, $\Gamma \vdash_{\mathrm{ST}} t : \kappa$, $\Theta :: \Gamma$, and $\bar{\theta} :: \kappa$ hold for some Γ and κ; for example, in (Var), it must be the case that $\tau :: \kappa$.

Many of the rules are the same as those of standard intersection type systems, but peculiar to our type system is the use of $\sqcup$ in the rules $(\wedge)$ and $(\mathbf{Y}1)$. In $(\wedge)$, the premises say that each t_i is used according to τ_i; think of t_i as a "used" part of some term t such that $t_i \sqsubseteq t$. In the conclusion, those used parts $t_1,\ldots,t_n$ are "merged" to obtain $\bigsqcup_{i\in[n]} t_i$ as the used part of t when it is accessed according to types $\tau_1,\ldots,\tau_n$. For example, consider the term $\lambda x^{\mathtt{o}\to\mathtt{o}\to\mathtt{o}}.\lambda y^{\mathtt{o}}.\lambda z^{\mathtt{o}}.x\,y\,z$. Then we have $\emptyset \vdash \lambda x^{\mathtt{o}\to\mathtt{o}\to\mathtt{o}}.\lambda y^{\mathtt{o}}.\lambda z^{\mathtt{o}}.x\,y\,\bot : (\mathtt{o} \to \top \to \mathtt{o}) \to$

$$\frac{}{x : \wedge\{\tau\} \vdash x^{\kappa} : \tau}(\mathrm{Var}) \quad \frac{\Theta, x : \theta \vdash t : \tau}{\Theta \vdash \lambda x.t : \theta \to \tau}(\mathrm{Abs1}) \quad \frac{\Theta \vdash t : \tau}{\Theta \vdash \lambda \bar{x}.t : \top \to \tau}(\mathrm{Abs2})$$

$$\frac{\Theta \vdash t : \theta \to \tau \;\; \Delta \vdash s : \theta}{\Theta \wedge \Delta \vdash t\, s : \tau}(\mathrm{App}) \quad \frac{\Theta \vdash t_1\,(\mathbf{Y} t_2) : \tau}{\Theta \vdash \mathbf{Y}(t_1 \sqcup t_2) : \tau}(\mathbf{Y}1) \quad \frac{\Theta \vdash t\,\bot : \tau}{\Theta \vdash \mathbf{Y} t : \tau}(\mathbf{Y}2)$$

$$\frac{\Theta_1 \vdash t_1 : \theta_1 \;\; \ldots \;\; \Theta_n \vdash t_n : \theta_n}{\bigwedge_{i\in[n]} \Theta_i \vdash a(t_1, \ldots, t_n) : \mathsf{o}}(a) \quad \frac{\Theta_1 \vdash t_1 : \tau_1 \;\; \ldots \;\; \Theta_n \vdash t_n : \tau_n}{\bigwedge_{i\in[n]} \Theta_i \vdash \bigsqcup_{i\in[n]} t_i : \bigwedge_{i\in[n]} \tau_i}(\wedge) \quad \frac{\Theta \vdash t : \bar{\theta}}{\Theta, x : \top \vdash t : \bar{\theta}}(\top)$$

Figure 2 The intersection type system for the minimality (see Section 3.1 for the operator $\sqcup$).

$\mathsf{o} \to \top \to \mathsf{o}$ and $\emptyset \vdash \lambda x^{\mathsf{o}\to\mathsf{o}\to\mathsf{o}}.\lambda y^{\mathsf{o}}.\lambda z^{\mathsf{o}}.x\,\bot\,z : (\top \to \mathsf{o} \to \mathsf{o}) \to \top \to \mathsf{o} \to \mathsf{o}$. From those judgments, we obtain

$$\emptyset \vdash \lambda x^{\mathsf{o}\to\mathsf{o}\to\mathsf{o}}.\lambda y^{\mathsf{o}}.\lambda z^{\mathsf{o}}.x\,y\,z : ((\mathsf{o} \to \top \to \mathsf{o}) \to \mathsf{o} \to \top \to \mathsf{o}) \wedge ((\top \to \mathsf{o} \to \mathsf{o}) \to \top \to \mathsf{o} \to \mathsf{o})$$

by using $(\wedge)$. Note that when $n = 0$, the rule $(\wedge)$ allows us to derive $\emptyset \vdash \bot : \top$.

There are two typing rules for $\mathbf{Y}\,t$. The rule $(\mathbf{Y}1)$ covers the case where $\mathbf{Y}\,t$ is reduced to $t(\mathbf{Y}\,t)$ and the argument $\mathbf{Y}\,t$ is used again; t_1 in the premise represents the used part of the head occurrence of t, whereas t_2 represents the used part of the occurrence of t in the argument $\mathbf{Y}\,t$. In the conclusion, both parts are merged to obtain $t_1 \sqcup t_2$ as the used part of t. For example, consider $\mathbf{Y}\,t$ where $t = \lambda f.\lambda x.\lambda y.\mathtt{b}\,x\,(f\,\bot\,y)$ and $\tau = \mathsf{o} \to \bot \to \mathsf{o}$. Then we have $\emptyset \vdash t_1(\mathbf{Y}\,t_2) : \tau$ for $t_1 = \lambda f.\lambda x.\lambda y.\mathtt{b}\,x\,(f\,\bot\,\bot)$ and $t_2 = \lambda f.\lambda x.\lambda y.\mathtt{b}\,\bot\,(f\,\bot\,\bot)$. By using $(\mathbf{Y}1)$, we obtain $\emptyset \vdash \mathbf{Y}(\lambda f.\lambda x.\lambda y.\mathtt{b}\,x\,(f\,\bot\,\bot)) : \tau$, which correctly models the used part of $\mathbf{Y}\,t$. The rule $(\mathbf{Y}2)$ is for the case where recursive calls do not contribute to the result. For example, consider the term $t = \mathbf{Y}(\lambda x.\mathtt{a}(\bot))$. Then from $\emptyset \vdash (\lambda x.\mathtt{a}(\bot))\bot : \mathsf{o}$, we obtain $\emptyset \vdash t : \mathsf{o}$.

The theorem below states that the minimality is correctly characterized by our intersection type system. See Appendix A for an outline of a proof; the full proof is found in [20].

▶ **Theorem 18** (soundness and completeness). *For every closed and ground-typed term t, t is minimal if and only if $\emptyset \vdash t : \bar{\theta}$ for some $\bar{\theta}$.*

We give examples of type derivations below.

▶ **Example 19** (cf. Example 10). Let $t = (\lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}})\;\mathtt{a}\;\bot^{\mathsf{o}}$ and $s = (\lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}})\;\mathtt{a}\;\mathtt{a}$. Then we can show that t is minimal by giving the derivation tree of $\emptyset \vdash t : \mathsf{o}$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \wedge\{\mathsf{o}\} \vdash x^{\mathsf{o}} : \mathsf{o}}(\mathrm{Var})}{x : \wedge\{\mathsf{o}\} \vdash \lambda y^{\mathsf{o}}.x^{\mathsf{o}} : \top \to \mathsf{o}}(\mathrm{Abs2})}{\emptyset \vdash \lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}} : \wedge\{\mathsf{o}\} \to \top \to \mathsf{o}}(\mathrm{Abs1}) \quad \dfrac{\dfrac{}{\emptyset \vdash \mathtt{a} : \mathsf{o}}(\mathtt{a})}{\emptyset \vdash \mathtt{a} : \wedge\{\mathsf{o}\}}(\wedge)}{\emptyset \vdash (\lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}})\;\mathtt{a} : \top \to \mathsf{o}}(\mathrm{App}) \quad \dfrac{}{\emptyset \vdash \bot^{\mathsf{o}} : \top}(\wedge)}{\emptyset \vdash (\lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}})\;\mathtt{a}\;\bot^{\mathsf{o}} : \mathsf{o}}(\mathrm{App})$$

In contrast, $\emptyset \nvdash s : \mathsf{o}$, because $x : \wedge\{\mathsf{o}\}, y : \wedge\{\mathsf{o}\} \nvdash x^{\mathsf{o}} : \mathsf{o}$.

The following is a more complex example, where intersection types play an important role.

▶ **Example 20.** Let $s = (\lambda f^{(\mathsf{o}\to\mathsf{o}\to\mathsf{o})\to\mathsf{o}}.\mathsf{a}(f\,\mathsf{fst}, f\,\mathsf{snd}))$, $u = (\lambda g^{\mathsf{o}\to\mathsf{o}\to\mathsf{o}}.g\,\mathsf{b}\,\mathsf{c})$, and $t = s\,u$, where $\mathsf{fst} = \lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.x^{\mathsf{o}}$ and $\mathsf{snd} = \lambda x^{\mathsf{o}}.\lambda y^{\mathsf{o}}.y^{\mathsf{o}}$. Then $\emptyset \vdash t : \mathsf{o}$ is derived from the following two derivations by applying (App), where $\tau_1 = \wedge\{\mathsf{o}\} \to \top \to \mathsf{o}$ and $\tau_2 = \top \to \wedge\{\mathsf{o}\} \to \mathsf{o}$. Hence this t is minimal.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : \wedge\{\wedge\{\tau_1\}\to\mathsf{o}\} \vdash f : \wedge\{\tau_1\}\to\mathsf{o}}(\mathrm{Var}) \quad \dfrac{\dfrac{\text{(similarly to Example 19)}}{\emptyset \vdash \mathsf{fst} : \tau_1}}{\emptyset \vdash \mathsf{fst} : \wedge\{\tau_1\}}(\wedge)}{f : \wedge\{\wedge\{\tau_1\}\to\mathsf{o}\} \vdash f\,\mathsf{fst} : \mathsf{o}}(\mathrm{App})}{f : \wedge\{\wedge\{\tau_1\}\to\mathsf{o}\} \vdash f\,\mathsf{fst} : \wedge\{\mathsf{o}\}}(\wedge) \quad \dfrac{\text{(similarly to the left)}}{f : \wedge\{\wedge\{\tau_2\}\to\mathsf{o}\} \vdash f\,\mathsf{snd} : \wedge\{\mathsf{o}\}}(\wedge)}{f : \wedge\{\wedge\{\tau_1\}\to\mathsf{o}, \wedge\{\tau_2\}\to\mathsf{o}\} \vdash \mathsf{a}(f\,\mathsf{fst}, f\,\mathsf{snd}) : \mathsf{o}}(\mathsf{a})}{\emptyset \vdash \lambda f.\mathsf{a}(f\,\mathsf{fst}, f\,\mathsf{snd}) : \bigwedge_{l\in[2]}\{\wedge\{\tau_l\}\to\mathsf{o}\}\to\mathsf{o}}(\mathrm{Abs1})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{g : \wedge\{\tau_1\} \vdash g : \wedge\{\mathsf{o}\}\to\top\to\mathsf{o}}(\mathrm{Var}) \quad \dfrac{\dfrac{}{\emptyset \vdash \mathsf{b} : \mathsf{o}}(\mathsf{b})}{\emptyset \vdash \mathsf{b} : \wedge\{\mathsf{o}\}}(\wedge)}{g : \wedge\{\tau_1\} \vdash g\,\mathsf{b} : \top\to\mathsf{o}}(\mathrm{App}) \quad \dfrac{}{\emptyset \vdash \bot : \top}(\wedge)}{g : \wedge\{\tau_1\} \vdash g\,\mathsf{b}\,\bot : \mathsf{o}}(\mathrm{App}) \quad \dfrac{\dfrac{\text{(similarly to the left)}}{g : \wedge\{\tau_2\} \vdash g\,\bot\,\mathsf{c} : \mathsf{o}}(\mathrm{App})}{\emptyset \vdash \lambda g.g\,\bot\,\mathsf{c} : \wedge\{\tau_2\}\to\mathsf{o}}(\mathrm{Abs1})}{\emptyset \vdash \lambda g.g\,\mathsf{b}\,\mathsf{c} : \bigwedge_{l\in[2]}\{\wedge\{\tau_l\}\to\mathsf{o}\}}(\wedge)}(\mathrm{Abs1})$$

Note that the term u is "used" in two different ways in t: in $f\,\mathsf{fst}$, the subterm b is used, whereas in $f\,\mathsf{snd}$, the subterm c is used.

6 Proof of the Main Lemma (Lemma 14)

In this section, we prove Lemma 14 by using the intersection type system from the previous section. Recall that we need to prove that if $E[C] \in \hat{\Lambda}_n(k, \iota, \xi)$, then there is a context $D \succeq t_{\mathrm{HARD},k}$ such that $E[D] \in \hat{\Lambda}_n(k, \iota, \xi)$. Thanks to the result of the previous section, $E[C] \in \hat{\Lambda}_n(k, \iota, \xi)$ implies that $E[C]$ is typable in the intersection type system. Thus, it suffices to construct D of the same size such that (i) C has "the same typing properties" as D, and (ii) D contains $t_{\mathrm{HARD},k}$. To this end, we first extend the notion of types to those of contexts (called *context-types*) in Section 6.1. We then show in Section 6.2 that we can indeed construct a context D that has the same context types as C, and prove Lemma 14.

6.1 Context-Types

For each affine-context C, we write $C \lhd_{\mathrm{ST}} \{\langle\Gamma'_1, \kappa'_1\rangle, \ldots, \langle\Gamma'_n, \kappa'_n\rangle\} \Rrightarrow \langle\Gamma, \kappa\rangle$ if there is a derivation tree of $\Gamma \vdash_{\mathrm{ST}} C[\mathtt{x}] : \kappa$ with the assumptions $\{\Gamma'_1 \vdash_{\mathrm{ST}} \mathtt{x} : \kappa'_1, \ldots, \Gamma'_n \vdash_{\mathrm{ST}} \mathtt{x} : \kappa'_n\}$, where n is at most one and $\mathtt{x}$ is a variable not occurring in C. Intuitively, it means that there is a derivation tree of $\Gamma' \vdash_{\mathrm{ST}} C : \kappa'$ with the assumptions $\{\Gamma'_1 \vdash_{\mathrm{ST}} [\,] : \kappa'_1, \ldots, \Gamma'_n \vdash_{\mathrm{ST}} [\,] : \kappa'_n\}$ (see also Example 21). We often write $t \lhd_{\mathrm{ST}} \tilde{\theta}$ for $t \lhd_{\mathrm{ST}} \emptyset \Rrightarrow \tilde{\theta}$. We use $\tilde{\kappa}$ to denote a pair $\langle\Gamma, \kappa\rangle$ and use $\tilde{\nu}$ to denote a $\{\langle\Gamma'_1, \kappa'_1\rangle, \ldots, \langle\Gamma'_n, \kappa'_n\rangle\} \Rrightarrow \langle\Gamma, \kappa\rangle$. Note that C is a term (resp. a linear-context) if $C \lhd_{\mathrm{ST}} \{\langle\Gamma'_1, \kappa'_1\rangle, \ldots, \langle\Gamma'_n, \kappa'_n\rangle\} \Rrightarrow \langle\Gamma, \kappa\rangle$ holds for $n = 0$ (resp. $n = 1$). Below we extend the notion of $\lhd_{\mathrm{ST}}$ to the intersection type system. The set of *(affine-)context-types*, ranged over by $\tilde{\mu}$, is defined as follows, where $n \geq 0$ and we may write $\tilde{\theta}^+$ for $\tilde{\theta}$ if $\tilde{\theta} \neq \emptyset$:

$$\tilde{\tau} ::= \langle\Theta, \tau\rangle \qquad \tilde{\theta} ::= \{\tilde{\tau}_1, \ldots, \tilde{\tau}_n\} \qquad \tilde{\pi} ::= \tilde{\tau} \mid \tilde{\theta}^+ \qquad \tilde{\mu} ::= \tilde{\theta} \Rrightarrow \tilde{\pi}.$$

For $\tilde{\mu}$, intuitively, $\tilde{\theta} \Rrightarrow \tilde{\tau}$ denotes the pair of the assumptions ($\tilde{\theta}$) and the conclusion ($\tilde{\tau}$) of a derivation tree, and $\tilde{\theta} \Rrightarrow \tilde{\theta}^+$ denotes the pair of the assumptions ($\tilde{\theta}$) and the conclusions

($\tilde{\theta}^+$) of one or more derivation trees. The *refinement relation* is defined as the least relation closed under the following rules, where $n \geq 0$:

$$\frac{\Theta :: \Gamma \qquad \tau :: \kappa}{\langle \Theta, \tau \rangle :: \langle \Gamma, \kappa \rangle} \qquad \frac{\tilde{\tau}_1 :: \langle \Gamma, \kappa \rangle \qquad \dots \qquad \tilde{\tau}_n :: \langle \Gamma, \kappa \rangle}{\{\tilde{\tau}_1, \dots, \tilde{\tau}_n\} :: \langle \Gamma, \kappa \rangle} \qquad \frac{\tilde{\theta}' :: \langle \Gamma', \kappa' \rangle \qquad \tilde{\pi} :: \langle \Gamma, \kappa \rangle}{\tilde{\theta}' \Rrightarrow \tilde{\pi} :: \langle \Gamma', \kappa' \rangle \Rrightarrow \langle \Gamma, \kappa \rangle}.$$

Henceforth we only consider context-types occurring in this refinement relation (so, we always make the assumptions that for each $\tilde{\theta}' \Rrightarrow \tilde{\theta}$, for some $\langle \Gamma, \kappa \rangle$ and $\langle \Gamma', \kappa' \rangle$, $\tilde{\theta} :: \langle \Gamma, \kappa \rangle$ and $\tilde{\theta}' :: \langle \Gamma', \kappa' \rangle$). For each affine-context C, we write $C \lhd \{\langle \Theta'_1, \tau'_1 \rangle, \dots, \langle \Theta'_n, \tau'_n \rangle\} \Rrightarrow \langle \Theta, \tau \rangle$ if there is a derivation tree of $\Theta \vdash C[\mathtt{x}] : \tau$ with the assumptions $\{\Theta'_1 \vdash \mathtt{x} : \tau'_1, \dots, \Theta'_n \vdash \mathtt{x} : \tau'_n\}$, where $\mathtt{x}$ is a variable not occurring in C. For $n \geq 1$, we write $(\bigsqcup_{i \in [n]} C_i) \lhd (\bigcup_{i \in [n]} \tilde{\theta}'_i) \Rrightarrow \{\tilde{\tau}_1, \dots, \tilde{\tau}_n\}$ if $C_i \lhd \tilde{\theta}'_i \Rrightarrow \tilde{\tau}_i$ for each $i \in [n]$. We often write $t \lhd \tilde{\theta}$ for $t \lhd \emptyset \Rrightarrow \tilde{\theta}$.

▶ **Example 21.** Let $C = (\lambda g^{\kappa_0}.[\,]\,\mathtt{b}\,\mathtt{c})$, where $\kappa_0 = \mathtt{o} \to \mathtt{o} \to \mathtt{o}$. Note that $C[g]$ is the term u in Example 20. Then, we have $C \lhd_{\mathrm{ST}} \{\langle (g : \kappa_0), \kappa_0 \rangle\} \Rrightarrow \langle \emptyset, \kappa_0 \to \mathtt{o} \rangle$ by $g : \kappa_0 \vdash_{\mathrm{ST}} g : \kappa_0$ and $\emptyset \vdash_{\mathrm{ST}} C[g] : \kappa_0 \to \mathtt{o}$. Also, we have $C \lhd \{\langle (g : \tau_1), \tau_1 \rangle, \langle (g : \tau_2), \tau_2 \rangle\} \Rrightarrow \langle \emptyset, \bigwedge_{l \in [2]} \{\wedge\{\tau_l\} \to \mathtt{o}\} \rangle$ by using the derivation tree in Example 20 with regarding g as a hole, where $\tau_1 = \wedge\{\mathtt{o}\} \to \top \to \mathtt{o}$ and $\tau_2 = \top \to \wedge\{\mathtt{o}\} \to \mathtt{o}$. Furthermore, we also have $C \lhd \{\langle (g : \tau_1), \tau_1 \rangle, \langle (g : \tau_2), \tau_2 \rangle\} \Rrightarrow \{\langle \emptyset, \wedge\{\tau_1\} \to \mathtt{o} \rangle, \langle \emptyset, \wedge\{\tau_2\} \to \mathtt{o} \rangle\}$. It is because C is the join of $C_1 = (\lambda g^{\kappa_0}.[\,]\,\mathtt{b}\,\bot^{\mathtt{o}})$ and $C_2 = (\lambda g^{\kappa_0}.[\,]\,\bot^{\mathtt{o}}\,\mathtt{c})$. Here, note that $C[g] = C_1[g] \sqcup C_2[g]$, $C_1 \lhd \{\langle (g : \tau_1), \tau_1 \rangle\} \Rrightarrow \langle \emptyset, \wedge\{\tau_1\} \to \mathtt{o} \rangle$, and $C_2 \lhd \{\langle (g : \tau_2), \tau_2 \rangle\} \Rrightarrow \langle \emptyset, \wedge\{\tau_2\} \to \mathtt{o} \rangle$.

Below we list a few properties (see Appendix B for the proofs).

▶ **Proposition 22** (substitution). *Suppose that C is a linear-context. If $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$ and $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$, then $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}$.*

▶ **Proposition 23** (inverse substitution). *Suppose that C is a linear-context. If $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}$, then $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$ and $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$ for some $\tilde{\theta}'$.*

These properties enable us to replace contexts preserving the minimality. For example, given $\emptyset \vdash C[D[t]] : \mathtt{o}$ (i.e., $C[D[t]]$ is minimal); then by Proposition 23, $C \lhd \tilde{\theta} \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$, $D \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$, and $t \lhd \tilde{\theta}'$ for some $\tilde{\theta}$ and $\tilde{\theta}'$; then by Proposition 22, $C[D'[t]] \lhd \{\langle \emptyset, \mathtt{o} \rangle\}$ (hence, $C[D'[t]]$ is also minimal) for each linear context $D' \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$.

In the following subsection, we will show in Lemma 25 that for the term $C[D[t]]$ in the above, if the size $|D|$ is sufficiently large, then one can choose D' as a term satisfying (i) $D' \succeq t_{\mathrm{HARD},k}$, and (ii) $|D'| = |D|$. Thus, from a term $C[D[t]] \in \hat{\Lambda}_n(k, \iota, \xi)$ such that $|D|$ is sufficiently large, one can construct a term $C[D'[t]]$ satisfying (i) $C[D'[t]] \succeq t_{\mathrm{HARD},k}$, and (ii) $|C[D'[t]]| = |C[D[t]]|$ and $C[D'[t]]$ is minimal (hence, $C[D'[t]] \in \hat{\Lambda}_n(k, \iota, \xi)$). This transformation method will help us to show Lemma 14.

6.2 Proof of Lemma 14

Here, we fix parameters k, ι, and ξ. W.l.o.g., in the following, we only consider terms, contexts, and environments having only variables in a fixed set $\mathcal{V}_\xi \triangleq \{z_1, \dots, z_\xi\}$ (of size ξ). We say that $\langle \Gamma, \kappa \rangle$ is *($\langle k, \iota, \xi \rangle$-)bounded* if $\max\{\mathtt{ord}(\kappa') \mid \kappa' \in \{\kappa\} \cup \mathrm{rng}(\Gamma)\} \leq k$ and $\max\{\mathtt{iar}(\kappa') \mid \kappa' \in \{\kappa\} \cup \mathrm{rng}(\Gamma)\} \leq \iota$; and that $\langle \Gamma', \kappa' \rangle \Rrightarrow \langle \Gamma, \kappa \rangle$ is *bounded* if both $\langle \Gamma', \kappa' \rangle$ and $\langle \Gamma, \kappa \rangle$ are; and that a context-type $\tilde{\mu}$ is *bounded* if the $\tilde{\nu}$ such that $\tilde{\mu} :: \tilde{\nu}$ is bounded. We also say that t is *bounded* if $\mathtt{ord}(t) \leq k$ and $\mathtt{iar}(t) \leq \iota$; and that a linear-context C is *bounded* if $C[\bot]$ is. Also, we use $\mathtt{a}$ (resp. $\mathtt{b}$, $\mathtt{c}$) to denote a tree constructor of arity 0 (resp. 2, 1).

The following technical lemma allows conversion between a ground-typed term and a term of a required typing property: see Appendix C for a proof.

▶ **Lemma 24.**

(1) *Suppose that* $\tilde{\theta}^+ :: \langle \Gamma, \kappa \rangle$ *is bounded. If* $\#(\mathrm{dom}(\Gamma)) < \xi$ *or* $\mathtt{ar}(\kappa) < \iota$*, then there exists a bounded linear-context* $C_{\tilde{\theta}^+}$ *such that* $C_{\tilde{\theta}^+} \lhd \{\langle \emptyset, \mathtt{o} \rangle\} \Rrightarrow \tilde{\theta}^+$.

(2) *Suppose that* $\tilde{\theta}$ *is bounded. Then, there exists a bounded affine-context* $D_{\tilde{\theta}}$ *such that* $D_{\tilde{\theta}} \lhd \tilde{\theta} \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$.

By Lemma 24, from a given bounded context-type $\tilde{\theta}' \Rrightarrow \tilde{\theta}^+$, one can construct a bounded affine-context having this context-type as $C_{\tilde{\theta}^+}[D_{\tilde{\theta}'}]$, except the case of that $\#(\mathrm{dom}(\Gamma)) = \xi$ and $\mathtt{ar}(\kappa) = \iota$. See Appendix C.1 for the boundary case; actually, terms having such context-type are of a special form.

The following is the key lemma, which shows that for any bounded context-type, one can construct a context D that has the context-type and contains the hard term $t_{\mathrm{HARD},k}$.

▶ **Lemma 25.** *Suppose that* $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}^+$ *for some bounded affine-context* C*. Then for some* m_0*, for every* $m \geq m_0$*, there exists a bounded affine-context* D *of size* m *such that* $D \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}^+$ *and* $D \succeq t_{\mathrm{HARD},k}$.

Proof. Let $\langle \Gamma, \kappa \rangle$ be such that $\tilde{\theta}^+ \lhd \langle \Gamma, \kappa \rangle$. Note that $\tilde{\theta}'$ and $\tilde{\theta}^+$ are also bounded.

(a) $\#(\mathrm{dom}(\Gamma)) < \xi$ or $\mathtt{ar}(\kappa) < \iota$: For each $l \geq 0$, let D_l be as follows, where $\mathtt{c}^l(\mathtt{a})$ is the term $\mathtt{c}(\ldots \mathtt{c}(\mathtt{a}) \ldots)$ that $\mathtt{c}$ occurs l times and $D_{\tilde{\theta}'}$ and $C_{\tilde{\theta}^+}$ are the ones in Lemma 24:

$$D_l \triangleq C_{\tilde{\theta}^+}[\mathtt{b}(t_{\mathrm{HARD},k}, \mathtt{b}(\mathtt{c}^l(\mathtt{a}), []))][D_{\tilde{\theta}'}].$$

Then $D_l \succeq t_{\mathrm{HARD},k}$ is obvious, and $D_l \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}^+$ by Proposition 22 (since $\mathtt{b}(t_{\mathrm{HARD},k}, \mathtt{b}(\mathtt{c}^l, [])) \lhd \{\langle \emptyset, \mathtt{o} \rangle\} \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$). Therefore, the claim has been proved by using these $D_1, D_2, \cdots$.

(b) Otherwise: Then, $\Gamma \vdash_{\mathrm{ST}} C[\bot] : \kappa$, $C[\bot]$ is bounded, and $\#(\mathrm{dom}(\Gamma)) = \xi$ and $\mathtt{ar}(\kappa) = \iota$, so C should be of the form $\lambda\_.C_0$ (see Appendix C.1). By Proposition 23, $C_0 \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}_0$ and $\lambda\_.[] \lhd \tilde{\theta}_0 \Rrightarrow \tilde{\theta}$ for some $\tilde{\theta}_0$. Then $\mathtt{ar}(C_0) < \mathtt{ar}(C) \leq \iota$ and $\tilde{\theta}_0 \neq \emptyset$ by $C_0 \neq \bot$ (since $\xi > 0$). Therefore by (a), for some m_0', there is $\{D_l'\}_{l \geq m_0'}$ such that $D_l' \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}_0$, $D_l' \succeq t_{\mathrm{HARD},k}$, and $|D_l'| = l$ for each $l \geq m_0'$. Let $D_l = \lambda\_.D_l'$. Then $D_l \succeq t_{\mathrm{HARD},k}$ is obvious, and $D_l \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}^+$ by Proposition 22. Therefore, the claim has been proved by using these $D_{m_0'}, D_{m_0'+1}, \cdots$. ◀

We are now ready to prove the main lemma.

Proof (of Lemma 14). Let $m \triangleq \max\{m_{\tilde{\theta}' \Rrightarrow \tilde{\theta}^+} \mid C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}^+ \text{ for some bounded } C\}$, where each $m_{\tilde{\theta}' \Rrightarrow \tilde{\theta}^+}$ is the m_0 in Lemma 25. Indeed such m exists, since the number of bounded context-types is finite. Recall $E[C] \in \hat{\Lambda}_n(k, \iota, \xi)$. Let $\tilde{E}$ be a linear-context such that $E[C] = \tilde{E}[C[t]]$ for some t or $E[C] = \tilde{E}[C]$. For the sake of brevity, we only write the case of that C is linear-context (i.e., $E[C] = \tilde{E}[C[t]]$). Since $\tilde{E}[C[t]]$ is minimal, $\emptyset \vdash \tilde{E}[C[t]] : \bar{\theta}$ for some $\bar{\theta} :: \mathtt{o}$ (Theorem 18). Then $\tilde{E}[C[t]] \lhd \emptyset \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$ (by $\tilde{E}[C[t]] \neq \bot$). By Proposition 23, there exist $\tilde{\theta}$ and $\tilde{\theta}'$ such that $\tilde{E} \lhd \tilde{\theta} \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$, $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$, and $t \lhd \emptyset \Rrightarrow \tilde{\theta}'$. By Lemma 25 (and $C \neq \bot$), there exists a bounded linear-context $D \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$ such that $D \succeq t_{\mathrm{HARD},k}$ and $|D| = |C|$. Therefore $\tilde{E}[D[t]] \lhd \emptyset \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$ (hence, $\emptyset \vdash \tilde{E}[D[t]] : \wedge\{\mathtt{o}\}$) by Proposition 22, and thus $E[D]$ is minimal (Theorem 18). Hence, $E[D] \in \hat{\Lambda}_n(k, \iota, \xi)$. ◀

7 Related Work

Ong [21] proved the k-EXPTIME completeness of higher-order model checking. There have also been results on parameterized complexity [15, 17, 18] and the complexity of subclasses of the problem [18, 5]. To our knowledge, however, they are all about the worst-case complexity. Despite the extremely high worst-case complexity, practical model checkers have

been developed that run quite fast for typical inputs [14, 4, 23, 28], which has led to the motivating question for our work: is higher-order model checking really hard in the average case?

Technically, closest to ours is the work of Asada et al. [26, 1] on a quantitative analysis of the length of β-reduction sequences of simply-typed λ-terms. In fact, our use of the tree-version of infinite monkey theorem (to show that almost every term contains a "hard" term), as well as the tree decomposition [1] has been inspired by their work and other studies on quantitative analysis of the λ-calculus and combinatory logics [8, 2]. The main new difficulty was that, unlike in the case of the length of β-reduction sequences, even if t is a "hard" term to model-check, a term $C[t]$ that contains t as a subterm may not be hard to model-check, because t may not actually be used in $C[t]$ or may be irrelevant for the property to be checked. This has led us to restrict terms to "minimal ones" that do not contain unnecessary subterms. The restriction turned out to be natural also for our goal: we wish to model the *average* case that arises in the actual applications to program verification, and the restriction to minimal terms helps us exclude out unlikely inputs.

We have used an intersection type system to characterize minimal terms. Related type systems have been studied in the context of useless code elimination [6, 7, 13]. In particular, Daminani [7] also used an intersection type system. To our knowledge, however, previous studies do not provide a *complete* characterization of minimal terms (especially in the presence of recursion).

There has been much interest in the average-case complexity in the field of computational complexity: see [3] for a good survey. In their terminology, our ultimate goal is to answer whether $(\mathrm{HOMC}_k(\cdot,\cdot),\mathcal{U})$ belongs to $\mathrm{Avg}_\delta\mathrm{DTIME}(f(n))$ (the class of distributional problems that can be solved in time $f(n)$ for at least $(1-\delta(n))$-fraction of the inputs of size n),[2] where $\mathrm{HOMC}_k(\cdot,\cdot)$ is the higher-order model checking problem of order k, $\mathcal{U}$ is a uniform distribution on inputs of each size n, δ is a function that is asymptotically smaller than $\lambda n.1$, and $f(n)$ is a function asymptotically much smaller than $exp_k(cn)$ (a k-fold exponential function). The result obtained in the present paper (Theorem 7) is not yet of this form, and is rather a mixture of average-case and worst-case analysis, which may be of independent interest from the perspective of complexity theory.

8 Conclusion

We have studied a mixture of average-case and worst-case complexity of higher-order model checking, and shown that for almost every minimal λY-term t of order-k, the higher-order model checking problem specialized for t is k-EXPTIME hard with respect to the size of a tree automaton. To our knowledge, this is the first result on the average-case hardness of higher-order model checking. To obtain the result, we have given a complete type-based characterization of "minimal" terms that contain no useless subterms, which may be of independent interest. Pure average-case analysis of the hardness of higher-order model checking is left for future work.

[2] A similar notion has also been studied under the name "generic-case complexity" [11].

References

1 Kazuyuki Asada, Naoki Kobayashi, Ryoma Sin'ya, and Takeshi Tsukada. Almost Every Simply Typed Lambda-Term Has a Long Beta-Reduction Sequence. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:16)2019.

2 Maciej Bendkowski, Katarzyna Grygiel, and Marek Zaionc. On the likelihood of normalization in combinatory logic. *J. Log. Comput.*, 27(7):2251–2269, 2017. doi:10.1093/logcom/exx005.

3 Andrej Bogdanov and Luca Trevisan. Average-case complexity. *CoRR*, abs/cs/0606037, 2006. arXiv:cs/0606037.

4 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Proceedings of Computer Science Logic (CSL)*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

5 Pierre Clairambault, Charles Grellois, and Andrzej S. Murawski. Linearity in higher-order recursion schemes. *PACMPL*, 2(POPL):39:1–39:29, 2018. doi:10.1145/3158127.

6 Mario Coppo, Ferruccio Damiani, and Paola Giannini. Refinement types for program analysis. In *Proceedings of International Static Analysis Symposium (SAS)*, volume 1145 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 1996. doi:10.1007/3-540-61739-6_39.

7 Ferruccio Damiani. A conjunctive type system for useless-code elimination. *Mathematical Structures in Computer Science*, 13(1):157–197, 2003. doi:10.1017/S0960129502003869.

8 René David, Katarzyna Grygiel, Jakub Kozik, Christophe Raffalli, Guillaume Theyssier, and Marek Zaionc. Asymptotically almost all λ-terms are strongly normalizing. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:2)2013.

9 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 1 edition, 2009. doi:10.1017/CBO9780511801655.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

11 Ilya Kapovich, Alexei G. Myasnikov, Paul Schupp, and Vladimir Shpilrain. Generic-case complexity, decision problems in group theory and random walks. *CoRR*, abs/math/0203239, 2002. URL: https://arxiv.org/abs/math/0203239.

12 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *Proceedings of International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

13 Naoki Kobayashi. Type-based useless-variable elimination. *Higher-Order and Symbolic Computation*, 14(2-3):221–260, 2001. doi:10.1023/A:1012944815270.

14 Naoki Kobayashi. Model-checking higher-order functions. In *Proceedings of ACM SIGPLAN conference on Principles and Practice of Declarative Programming (PPDP)*, pages 25–36. ACM Press, 2009. doi:10.1145/1599410.1599415.

15 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 416–428. ACM Press, 2009. doi:10.1145/1594834.1480933.

16 Naoki Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013. doi:10.1145/2487241.2487246.

17 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science (LICS)*, pages 179–188. IEEE Computer Society Press, 2009. doi:10.1109/LICS.2009.29.

18 Naoki Kobayashi and C.-H. Luke Ong. Complexity of Model Checking Recursion Schemes for Fragments of the Modal Mu-Calculus. *Logical Methods in Computer Science*, 7(4), 2012. doi:10.2168/LMCS-7(4:9)2011.

19 Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 222–233. ACM Press, 2011. doi:10.1145/1993498.1993525.

20 Yoshiki Nakamura, Kazuyuki Asada, Naoki Kobayashi, Ryoma Sin'ya, and Takeshi Tsukada. On average-case hardness of higher-order model checking. **A full version.** Available from https://www.kb.is.s.u-tokyo.ac.jp/~koba/papers/OnAverageCaseHOMC.pdf.

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

22 C.-H. Luke Ong and Steven Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 587–598. ACM Press, 2011. doi:10.1145/1925844.1926453.

23 Steven Ramsay, Robin Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 61–72. ACM Press, 2014. doi:10.1145/2535838.2535873.

24 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In *Proceedings of International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011. doi:10.1007/978-3-642-22012-8_12.

25 Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *Proceedings of International Workshop on Reachability Problems (RP)*, volume 7550 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2012. doi:10.1007/978-3-642-33512-9_2.

26 Ryoma Sin'ya, Kazuyuki Asada, Naoki Kobayashi, and Takeshi Tsukada. Almost every simply typed λ-term has a long β-reduction sequence. In *Proceedings of International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 10203 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2017. doi:10.1007/978-3-662-54458-7_4.

27 Richard Statman. On the lambda Y calculus. *APAL*, 130(1-3):325–337, 2004. doi:10.1016/j.apal.2004.04.004.

28 Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada. Streett automata model checking of higher-order recursion schemes. In *Proceedings of International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 84 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.32.

A Supplementary Materials for Section 5

This section sketches a proof of Theorem 18 in Section 5, as well as that of Proposition 11. The full proofs can be found in the long version [20].

A.1 A Characterization of the Minimality

We give a technically convenient characterization of the minimality. Let t be a closed and ground-typed term and u be a subterm of t, i.e. $t = C[u]$ for some linear-context C. Recall that t is minimal if it has no useless subterm; so in particular u must be "used". Intuitively

u in $C[u]$ is used if $t = C[u] \longrightarrow^* E[u]$ for some evaluation context E,[3] but this criterion is incorrect. Consider, for example,

$$t \quad = \quad a\big((\lambda x.x)\, u,\ \bot\, \underline{u}\big) \quad \longrightarrow \quad a(u,\ \bot u),$$

where u appears under an evaluation context $E = a([\,], \bot\, u)$ after the reduction but the underlined occurrence of u is indeed useless. This example suggests that we need to distinguish different occurrences of the same term.

Let ℓ be a special tree constructor of arity 1 such that $\ell \notin \Sigma$. We call this symbol a *label* and use it to mark focused occurrences of (sub)terms. The *labelling* operation $(-)^\ell$ is defined, for a term t of type $\kappa_1 \to \ldots \to \kappa_k \to \mathsf{o}$, by $t^\ell := \lambda z_1 \ldots \lambda z_k.\ell(t\, z_1\, \ldots\, z_k)$. For a term t possibly having ℓ, we write $\natural(t)$ for the term obtained by removing ℓ.

A term is *labelled* if it contains ℓ; otherwise it is *unlabelled*. A labelled finite tree V is *well-labelled* if $V = D[\ell(u)]$ and $\natural(u) \neq \bot$ for some D and u.

▶ **Theorem 26** (Characterization of the minimality). *Let t be a closed and ground-typed term over Σ. Then, t is minimal if and only if for every $\langle C, s\rangle$ such that $t = C[s]$ and $s \neq \bot$, there exists a finite well-labelled tree V such that $C[s^\ell] \longrightarrow^* \sqsupseteq V$.*

A.2 Proof Sketch of Proposition 11

Let t be a closed and ground-typed term. Assume that t is minimal, and let s be a non-$\bot$, closed and ground-typed subterm of t. Then $t = C[s]$ for some linear-context C. By Theorem 26, the value tree of $C[s^\ell]$ contains ℓ, say $T(C[s^\ell]) = D[\ell(V)]$. One can show that V is the value tree of s, which implies that $T(C[s^\ell])$ contains $T(s)$ as a subtree. Since $T(t) = T(C[s])$ is obtained by removing ℓ from $T(C[s^\ell])$, it also contains $T(s)$ as a subtree.

A.3 Proof Sketch of Theorem 18

Since we shall study possibly labelled terms, we first introduce a typing rule for $\ell(t)$:

$$\frac{\Theta \vdash t : \mathsf{o}}{\Theta \vdash \ell(t) : \mathsf{o}}(\ell) \quad .$$

Note that the rule for $\ell(t)$ differs from that for tree constructors: the argument of ℓ must be of type o, whereas those of a tree constructor can be of type $\bot$ in addition to o. The above rule ensures that $\emptyset \vdash \ell(t) : \mathsf{o}$ implies $\natural(t) \neq \bot$. So we have the following lemma.

▶ **Lemma 27.** *Let V be a labelled finite tree. If $\emptyset \vdash V : \mathsf{o}$, then V is well-labelled.*

We use Subject Reduction and Subject Expansion in the soundness and completeness proofs of our system, similar to proofs for other intersection type systems. However the standard version of Subject Reduction and Subject Expansion does not hold for our system since minimality is not preserved by reduction nor expansion. For example, consider

$$\Big(\lambda f.a\big(f\,(\lambda x\_.x),\ f\,(\lambda \_y.y)\big)\Big)\,(\lambda g.g\,b\,c) \quad \longrightarrow \quad a\big((\lambda g.g\,b\,c)\,(\lambda x\_.x),\ (\lambda g.g\,b\,c)\,(\lambda \_y.y)\big),$$

where a, b and c are tree constructors; the left-hand-side is minimal but the right-hand-side is not. In order to retain minimality, the right-hand-side has to be approximated:

$$a\big((\lambda g.g\,b\,c)\,(\lambda x\_.x),\ (\lambda g.g\,b\,c)\,(\lambda \_y.y)\big) \quad \sqsupseteq \quad a\big((\lambda g.g\,b\,\bot)\,(\lambda x\_.x),\ (\lambda g.g\,\bot\,c)\,(\lambda \_y.y)\big).$$

[3] Although evaluation contexts are not explicitly defined in Section 2, they are implicitly given in Definition 2 and their concrete definition should be clear.

The next lemma is a correct version, which takes account of the approximation relation.

▶ **Lemma 28** (Subject Reduction / Subject Expansion). *Assume that $s_1 \sqsubseteq t_1$ and $\Theta \vdash s_1 : \bar{\theta}$.*
(1) *If $t_1 \longrightarrow^* t_2$, then there exists $s_2 \sqsubseteq t_2$ with $\Theta \vdash s_2 : \bar{\theta}$ such that $s_1 \longrightarrow^* \sqsupseteq s_2$. Furthermore, if s_1 is labelled, we can choose s_2 so that it is labelled.*
(2) *If $t_0 \longrightarrow^* t_1$, then there exists $s_0 \sqsubseteq t_0$ with $\Theta \vdash s_0 : \bar{\theta}$ such that $s_0 \longrightarrow^* \sqsupseteq s_1$.*

The proof of completeness is rather straightforward. Note that, given a term t and a tree V with $t \longrightarrow^* \sqsupseteq V$, Subject Expansion induces a derivation of $\emptyset \vdash t' : \bar{\theta}$ for some $t' \sqsubseteq t$. The key to the completeness proof is to find sufficiently large V so that $t' = t$.

▶ **Theorem 29** (completeness). *Let t be any closed and ground-typed term over Σ. If t is minimal, then $\emptyset \vdash t : \bar{\theta}$ for some $\bar{\theta}$.*

Proof sketch. Since t is minimal, by Theorem 26, for each $\langle C, s\rangle$ such that $t = C[s]$ and $s \neq \bot$, one can find a finite well-labelled tree $V_C = D_C[\ell(u_C)]$ such that

$$C[s^\ell] \quad \longrightarrow^* \sqsupseteq \quad V_C \quad = \quad D_C[\ell(u_C)]. \tag{1}$$

We can assume without loss of generality that ℓ does not occur in D_C. Let $V = \bigsqcup_C D_C[\natural u_C]$, where C ranges over linear contexts such that $t = C[s]$ holds for some $s \neq \bot$. This is well-defined since $D_C[\natural u_C] \sqsubseteq T(t)$ for every C. Since V is an unlabelled tree, $\emptyset \vdash V : \bar{\theta}$ for some $\bar{\theta}$. Then by Subject Expansion (Lemma 28(2)), there exists $t' \sqsubseteq t$ such that $t' \longrightarrow^* \sqsupseteq V$ and $\emptyset \vdash t' : \bar{\theta}$.

It suffices to show that $t' = t$. Assume $t' \sqsubset t$ for contradiction. By the assumption, there exists $\langle C, s\rangle$ such that $t = C[s]$, $s \neq \bot$, and $t' \sqsubseteq C[\bot]$. Then

$$C[s^\ell] \quad \sqsupseteq \quad C[\bot] \quad \sqsupseteq \quad t' \quad \longrightarrow^* \sqsupseteq \quad V \quad \sqsupseteq \quad D_C[\natural u_C],$$

and thus $C[s^\ell] \longrightarrow^* \sqsupseteq D_C[\natural u_C]$. This together with (1) implies that $D_C[\ell(u_C)] \sqcup D_C[\natural u_C]$ is well-defined. This means $\ell(u_C) \sqcup \natural(u_C)$ is well-defined, which contradicts to the assumption that $\natural(u_C) \neq \bot$. ◀

The soundness proof requires another trick, since Subject Reduction alone does not ensure that a label eventually appears under an evaluation context in the presence of divergence. A term is $\mathbf{Y}$-*free* if it does not have $\mathbf{Y}$. The evaluation of a $\mathbf{Y}$-free term always terminates, and the soundness of the type system for $\mathbf{Y}$-free terms is relatively easy to prove. So we aim to remove $\mathbf{Y}$ in a given term before applying Subject Reduction, preserving its type.

Consider the rewriting rule $C[\mathbf{Y}\, t] \hookrightarrow C[t\,(\mathbf{Y}\, t)]$, which is allowed to be applied to any occurrence of $\mathbf{Y}\, t$ (not restricted to those under evaluation contexts). Then $\succeq_{\mathbf{Y}}$ is defined as $\hookrightarrow^* \sqsupseteq$. The next lemma is a key to the soundness proof, reflecting the inductive nature of the rules for $\mathbf{Y}$ in our type system.

▶ **Lemma 30.** *Assume that $\Theta \vdash t : \bar{\theta}$. Then there exists a $\mathbf{Y}$-free term s such that $t \succeq_{\mathbf{Y}} s$ and $\Theta \vdash s : \bar{\theta}$. Furthermore, if t is labelled, one can choose s so that s is also labelled.*

▶ **Theorem 31** (soundness). *Let t be any closed and ground-typed term over Σ. If $\emptyset \vdash t : \bar{\theta}$ for some $\bar{\theta}$, then t is minimal.*

Proof sketch. If $\bar{\theta} = \top$, then $t = \bot$ and thus t is minimal. Otherwise, we can assume without loss of generality that $\bar{\theta} = \mathsf{o}$. By Theorem 26, it suffices to show that, for every $\langle C, s\rangle$ with $t = C[s]$ and $s \neq \bot$, there exists a finite well-labelled tree V such that $C[s^\ell] \longrightarrow^* \sqsupseteq V$.

Assume that $t = C[s]$ and $s \neq \bot$. Since $\emptyset \vdash C[s] : \mathtt{o}$ and $s \neq \bot$, one can show that $\emptyset \vdash C[s^\ell] : \mathtt{o}$. By Lemma 30, there exists a $\mathbf{Y}$-free labelled term $\emptyset \vdash u : \mathtt{o}$ such that $C[s^\ell] \succeq_{\mathbf{Y}} u$. Since u is $\mathbf{Y}$-free, its evaluation terminates, i.e. $u \longrightarrow^* W$ for some tree W. By Subject Reduction (Lemma 28(1)), there exists a labelled term $V \sqsubseteq W$ such that $u \longrightarrow^* \sqsupseteq V$ and $\emptyset \vdash V : \mathtt{o}$.

It suffices to show that $C[s^\ell] \longrightarrow^* \sqsupseteq V$ and that V is a well-labelled tree. The former claim follows from $C[s^\ell] \succeq_{\mathbf{Y}} u \longrightarrow^* \sqsupseteq V$ because $\succeq_{\mathbf{Y}}$ can be seen as a kind of reduction. To prove the latter, observe that V is a tree since every approximation of a tree is a tree. So V as a well-typed and labelled tree is well-labelled by Lemma 27. ◀

B Proof of Proposition 22 and 23

▶ **Lemma 32.** *Suppose that C is a linear-context. If $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\tau}$ and $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$, then $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \{\tilde{\tau}\}$.*

Proof. Let $\tilde{\theta}' = \{\tilde{\tau}'_1, \dots, \tilde{\tau}'_n\}$. By $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$, there exists $\{\langle \tilde{\theta}''_{i,j}, C'_{i,j}\rangle\}_{i\in[n], j\in[k_i]}$ such that $C' = \bigsqcup_{i\in[n],j\in[k_i]} C'_{i,j}$, $\tilde{\theta}'' = \bigcup_{i\in[n],j\in[k_i]} \tilde{\theta}''_{i,j}$, and $C'_{i,j} \lhd \tilde{\theta}''_{i,j} \Rrightarrow \tilde{\tau}'_i$. Here, we can assume that $k_1 = \cdots = k_n$ (so, we denote them by k). Then from the derivation tree of $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\tau}$ (see the left-hand side below), we can construct a derivation tree of $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\tau}$ (see the right-hand side below) by copying the form of the derivation tree of $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\tau}$ as follows, where $\tilde{\tau} = \langle \Theta, \tau\rangle$ and $f\colon [m] \to [n']$ is a surjective map:

$$\dfrac{\mathtt{x} \lhd \tilde{\tau}'_{f(1)} \quad \cdots \quad \mathtt{x} \lhd \tilde{\tau}'_{f(m)}}{\begin{array}{c}\vdots\,\mathcal{T}\\ \Theta \vdash C[\mathtt{x}] : \tau\end{array}} \rightsquigarrow_{\mathrm{copy}\ \mathcal{T}} \dfrac{\dfrac{C'_{f(1),1} \lhd \tilde{\theta}''_{f(1),1} \Rrightarrow \tilde{\tau}'_{f(1)} \quad \cdots \quad C'_{f(m),1} \lhd \tilde{\theta}''_{f(m),1} \Rrightarrow \tilde{\tau}'_{f(m)}}{\begin{array}{c}\vdots\,\mathcal{T}\\ \Theta \vdash C[\bigsqcup_{i\in[n]} C'_{i,1}] : \tau\end{array}} \quad \cdots \quad \dfrac{\cdots \quad \cdots \quad \cdots}{\begin{array}{c}\vdots\,\mathcal{T}\\ \Theta \vdash C[\bigsqcup_{i\in[n]} C'_{i,k}] : \tau\end{array}}}{\Theta \vdash C[C'] : \tau}\,(\wedge) \quad .$$

◀

Proof of Proposition 22. Let $\tilde{\theta}' = \{\tilde{\tau}'_1, \dots, \tilde{\tau}'_{n'}\}$ and $\tilde{\theta} = \{\tilde{\tau}_1, \dots, \tilde{\tau}_n\}$. By $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$, there exists $\{\langle \tilde{\theta}'_i, C_i\rangle\}_{i\in[m]}$ such that $C = \bigsqcup_{i\in[m]} C_i$, $\tilde{\theta}' = \bigcup_{i\in[m]} \tilde{\theta}'_i$, and $C_i \lhd \tilde{\theta}'_i \Rrightarrow \tilde{\tau}_{f(i)}$. By $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$, there exists $\{\langle \tilde{\theta}''_j, C''_j\rangle\}_{j\in[n']}$ such that $C' = \bigsqcup_{j\in[n']} C''_j$, $\tilde{\theta}'' = \bigcup_{j\in[n']} \tilde{\theta}''_j$, and $C''_j \lhd \tilde{\theta}''_j \Rrightarrow \{\tilde{\tau}_j\}$. Let $C'_i = \bigsqcup_{j\in[n'];\tilde{\tau}'_j\in\tilde{\theta}'_i} C''_j$ and let $\tilde{\theta}''_i = \bigcup_{j\in[n'];\tilde{\tau}'_j\in\tilde{\theta}'_i} \tilde{\theta}''_j$. Then $C'_i \lhd \tilde{\theta}''_i \Rrightarrow \tilde{\theta}'_i$. By Lemma 32, $C_i[C'_i] \lhd \tilde{\theta}''_i \Rrightarrow \tilde{\tau}_{f(i)}$. Therefore, $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}$. ◀

▶ **Lemma 33.** *Suppose that C is a linear-context. If $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\tau}$, then $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\tau}$ and $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$ for some $\tilde{\theta}'$.*

Proof. Then (the derivation tree of) $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\tau}$ should be of the form in the right-hand side below, where $\tilde{\tau} = \langle \Theta, \tau\rangle$, $C' = \bigsqcup_{i\in[m]} C'_i$, and $\tilde{\theta}'' = \bigcup_{i\in[m]} \tilde{\theta}''_i$. We let $\tilde{\theta}' = \{\tilde{\tau}'_1, \dots, \tilde{\tau}'_m\}$. Then, $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$ is immediate and $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\tau}$ is shown by replacing each subterm arise from t to $\mathtt{x}$ (see the left-hand side below):

$$\dfrac{\mathtt{x} \lhd \tilde{\tau}'_1 \quad \cdots \quad \mathtt{x} \lhd \tilde{\tau}'_m}{\begin{array}{c}\vdots\,\mathcal{T}\\ \Theta \vdash C[\mathtt{x}] : \tau\end{array}} \rightsquigarrow \dfrac{C'_1 \lhd \tilde{\theta}''_1 \Rrightarrow \tilde{\tau}'_1 \quad \cdots \quad C'_m \lhd \tilde{\theta}''_m \Rrightarrow \tilde{\tau}'_m}{\begin{array}{c}\vdots\,\mathcal{T}\\ \Theta \vdash C[C'] : \tau\end{array}} \quad .$$

◀

Proof of Proposition 23. Let $\tilde{\theta}'' = \{\tilde{\tau}''_1, \dots, \tilde{\tau}''_{n''}\}$ and $\tilde{\theta} = \{\tilde{\tau}_1, \dots, \tilde{\tau}_n\}$. By $C[C'] \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}$, there exist a surjective map $f\colon [m] \to [n]$ and a sequence $\{\langle C_i, C'_i, \tilde{\theta}''_i\rangle\}_{i\in[m]}$ such that $C_i[C'_i] \lhd \tilde{\theta}''_i \Rrightarrow \tilde{\tau}_{f(i)}$, $C = \bigsqcup_{i\in[m]} C_i$, $C' = \bigsqcup_{i\in[m]} C'_i$, and $\tilde{\theta}'' = \bigcup_{i\in[m]} \tilde{\theta}''_i$ (see also the full version [20]). By Lemma 33, $C_i \lhd \tilde{\theta}'_i \Rrightarrow \tilde{\tau}_{f(i)}$ and $C'_i \lhd \tilde{\theta}''_i \Rrightarrow \tilde{\theta}'_i$ for some $\tilde{\theta}'_i$. We now let $\tilde{\theta}' = \bigcup_{j\in[m]} \tilde{\theta}'_i$. Then, both $C' \lhd \tilde{\theta}'' \Rrightarrow \tilde{\theta}'$ and $C \lhd \tilde{\theta}' \Rrightarrow \tilde{\theta}$ are immediate. ◀

C Proof of Lemma 24

The *size* of a simple type κ and a simple type environment Γ, written $|\kappa|$ and $|\Gamma|$ respectively, is defined by: $|\kappa| \triangleq 1$ if $\kappa = \mathtt{o}$, $|\kappa| \triangleq 1+|\kappa_1|+|\kappa_2|$ if $\kappa = \kappa_1 \to \kappa_2$, and $|\Gamma| \triangleq 1+\sum_{x\in\mathrm{dom}(\Gamma)} |\Gamma(x)|$.

▶ **Definition 34.** *The term $t_{\Gamma,\kappa}$ is inductively defined as follows, where in the second case, $l = \min\{i \in [\xi] \mid z_i \in \mathrm{dom}(\Gamma)\}$; and in the third case, $l = \min\{i \in [\xi] \mid z_i \notin \mathrm{dom}(\Gamma)\}$:*

$$t_{\Gamma,\kappa} \triangleq \begin{cases} \mathtt{a} & (\kappa = \mathtt{o} \text{ and } \Gamma = \emptyset) \\ \mathtt{b}(z_l t_{\emptyset,\kappa^1} \dots t_{\emptyset,\kappa^m},\, t_{\Gamma',\mathtt{o}}) & (\kappa = \mathtt{o} \text{ and } \Gamma = (\Gamma', z_l : \kappa^1 \to \dots \to \kappa^m \to \mathtt{o})) \\ \lambda z_l.t_{(\Gamma, z_l:\kappa'),\kappa''} & (\kappa = \kappa' \to \kappa'' \text{ and } \#(\mathrm{dom}(\Gamma)) < \xi) \\ (\lambda z_1.t_{(z_1:\mathtt{o}),\kappa})\ t_{\Gamma,\mathtt{o}} & (\kappa = \kappa' \to \kappa'' \text{ and } \mathtt{ar}\,(\kappa) < \iota) \\ \textit{undefined} & (\textit{otherwise}) \end{cases}.$$

▶ **Proposition 35.** *Suppose that $\langle\Gamma,\kappa\rangle$ is ($\langle k,\iota,\xi\rangle$-)bounded. If $\#(\mathrm{dom}(\Gamma)) < \xi$ or $\mathtt{ar}\,(\kappa) < \iota$, then (1) $t_{\Gamma,\kappa}$ is defined, (2) $\Gamma \vdash_{\mathrm{ST}} t_{\Gamma,\kappa} : \kappa$, and (3) $t_{\Gamma,\kappa}$ is bounded.*

Proof. By a straightforward induction on the parameter $\langle|\kappa|,|\Gamma|\rangle$. ◀

We now extend the above for intersection types.

▶ **Definition 36.** *The term $t_{\Theta,\bar\theta}$ is inductively defined as follows, where in the second case, $l = \min\{i \in [\xi] \mid z_i \in \mathrm{dom}(\Theta)\}$; and in the third case, $l = \min\{i \in [\xi] \mid z_i \notin \mathrm{dom}(\Theta)\}$:*

$$t_{\Theta,\bar\theta} \triangleq \begin{cases} \mathtt{a} & (\bar\theta = \mathtt{o} \text{ and } \Theta = \emptyset) \\ \mathtt{b}(\bigsqcup_{i\in[n]} z_l t_{\emptyset,\theta_i^1} \dots t_{\emptyset,\theta_i^m},\, t_{\Theta',\mathtt{o}}) & (\bar\theta = \mathtt{o} \text{ and } \Theta = (\Theta', z_l : \bigwedge_{i\in[n]} \theta_i^1 \to \dots \to \theta_i^m \to \mathtt{o})) \\ \lambda z_l.t_{(\Theta, z_l:\theta'),\tau''} & (\bar\theta = \theta' \to \tau'' \text{ and } \#(\mathrm{dom}(\Theta)) < \xi) \\ (\lambda z_1.t_{(z_1:\wedge\{\mathtt{o}\}),\bar\theta})\ t_{\Theta,\mathtt{o}} & (\bar\theta = \theta' \to \tau'' \text{ and } \mathtt{ar}\,(\kappa) < \iota) \\ \bigsqcup_{i\in[n]} t_{\Theta,\tau_i} & (\bar\theta = \bigwedge_{i\in[n]} \tau_i \text{ and } n \geq 1) \\ \bot^\kappa & (\bar\theta = \top^\kappa \text{ and } \Theta = \emptyset) \\ \textit{undefined} & (\textit{otherwise}) \end{cases}.$$

▶ **Proposition 37.** *Suppose that $\langle\Theta,\bar\theta\rangle :: \langle\Gamma,\kappa\rangle$ for some bounded $\langle\Gamma,\kappa\rangle$. If $\#(\mathrm{dom}(\Gamma)) < \xi$, $\mathtt{ar}\,(\kappa) < \iota$, or $\langle\Theta,\bar\theta\rangle = \langle\emptyset,\top\rangle$, then (1) $t_{\Theta,\bar\theta}$ is defined, (2) $t_{\Theta,\bar\theta} \sqsubseteq t_{\Gamma,\kappa}$, (3) $\Theta \vdash t_{\Theta,\bar\theta} : \bar\theta$, and (4) $t_{\Theta,\bar\theta}$ is bounded.*

Proof. By a straightforward induction on the parameter $\langle|\kappa|,|\Gamma|\rangle$. The existence of the join in each case can be ensured by the assumption (2). ◀

We now extend the above for context-types to prove Lemma 24.

▶ **Definition 38.** *The linear-context $C_{\tilde\tau}$ is inductively defined as follows, where in the second case, $l = \min\{i \in [\xi] \mid z_i \notin \mathrm{dom}(\Theta)\}$:*

$$C_{\langle\Theta,\tau\rangle} \triangleq \begin{cases} \mathtt{b}(t_{\Theta,\mathtt{o}}, [\,]) & (\tau = \mathtt{o}) \\ \lambda z_l.C_{\langle(\Theta, z_l:\theta'),\tau''\rangle} & (\tau = \theta' \to \tau'' \text{ and } \#(\mathrm{dom}(\Theta)) < \xi) \\ (\lambda z_1.t_{(z_1:\wedge\{\mathtt{o}\}),\tau})\ C_{\langle\Theta,\mathtt{o}\rangle} & (\tau = \theta' \to \tau'' \text{ and } \mathtt{ar}\,(\tau) < \iota) \\ \textit{undefined} & (\textit{otherwise}) \end{cases}.$$

For each $\tilde\theta^+ = \{\tilde\tau_1,\dots,\tilde\tau_n\}$, let $C_{\tilde\theta^+} \triangleq \bigsqcup_{i\in[n]} C_{\tilde\tau_i}$. This is well-defined thanks to Proposition 37(2).

▶ **Proposition 39.** *Suppose that $\tilde\theta^+ :: \langle\Gamma,\kappa\rangle$ for some bounded $\langle\Gamma,\kappa\rangle$. If $\#(\mathrm{dom}(\Gamma)) < \xi$ or $\mathtt{ar}\,(\kappa) < \iota$, then (1) $C_{\tilde\theta^+}$ is defined, (2) $C_{\tilde\theta^+} \lhd \{\langle\emptyset,\mathtt{o}\rangle\} \Rrightarrow \tilde\theta$, and (3) $C_{\tilde\theta^+}$ is bounded.*

Proof. By a straightforward induction on the parameter $\langle |\kappa|, |\Gamma| \rangle$. ◀

▶ **Definition 40.** *The linear-context $D_{\tilde{\tau}}$ is defined as follows, where in the first case, $l = \min\{i \in [\xi] \mid z_i \in \mathrm{dom}(\Theta)\}$; and in the second case, $\tau = \theta^1 \to \ldots \to \theta^m \to \mathtt{o}$:*

$$D_{\langle \Theta, \tau \rangle} \triangleq \begin{cases} (\lambda z_l . D_{\langle \Theta', \tau \rangle})\, t_{\emptyset, \theta_l} & (\Theta = (\Theta', z_l : \theta_l)) \\ \mathtt{c}([\,]\, t_{\emptyset,\theta^1}\ \ldots\ t_{\emptyset,\theta^m}) & (\Theta = \emptyset) \end{cases}.$$

Let $D_{\tilde{\theta}^+} \triangleq \bigsqcup_{i \in [n]} D_{\tilde{\tau}_i}$ for each $\tilde{\theta}^+ = \{\tilde{\tau}_1, \ldots, \tilde{\tau}_n\}$. This is well-defined thanks to Proposition 37(2). Also, specially, let $D_\emptyset \triangleq \mathtt{a}$.

▶ **Proposition 41.** *Suppose that $\tilde{\theta} :: \langle \Gamma, \kappa \rangle$ for some bounded $\langle \Gamma, \kappa \rangle$. Then, (1) $D_{\tilde{\theta}}$ is defined, (2) $D_{\tilde{\theta}} \lhd \tilde{\theta} \Rrightarrow \{\langle \emptyset, \mathtt{o} \rangle\}$, and (3) $D_{\tilde{\theta}}$ is bounded.*

Proof. By a straightforward induction on the parameter $\langle |\kappa|, |\Gamma| \rangle$. ◀

As a consequence of Proposition 39 and 41, Lemma 24 has been proved.

C.1 On the Boundary Case of Lemma 24(1)

Here, we consider the boundary case of Lemma 24(1), i.e., $\Gamma \vdash_{\mathrm{ST}} t : \kappa$, t is $\langle k, \iota, \xi \rangle$-bounded, $\#(\mathrm{dom}(\Gamma)) = \xi$, and $\mathtt{ar}\,(\kappa) = \iota$. Actually in this case, t should be of a special form.

▶ **Lemma 42.** *Suppose that*
(1) $\Gamma \vdash_{\mathrm{ST}} t : \kappa$,
(2) *t is $\langle k, \iota, \xi \rangle$-bounded,*
(3) $\#(\mathrm{dom}(\Gamma)) = \xi$, *and*
(4) $\mathtt{ar}\,(\kappa) = \iota$.
Then, t is α-equivalent to a term of the form $\lambda\_.t_1$.

Proof. By $\xi > 1$, $t \neq x$ and $t \neq \bot$. By $\iota > 0$, $t \neq a(t_1, \ldots, t_{\Sigma(a)})$. By $\mathtt{ar}\,(\kappa) = \iota$, $t \neq t_1 t_2$ and $t \neq \mathbf{Y} t_1$. Therefore t is of the form $\lambda \bar{x}.t_1$. By that t is bounded and $\#(\mathrm{dom}(\Gamma)) = \xi$, the last rule of $\Gamma \vdash_{\mathrm{ST}} \lambda \bar{x}.t_1 : \kappa$ should be (Abs2), so $\Gamma \vdash_{\mathrm{ST}} t_1 : \kappa''$, where $\kappa = \kappa' \to \kappa''$. Then $\bar{x}$ does not occur in t_1 as a free variable. Therefore t is α-equivalent to the term $\lambda\_.t_1$. ◀