Complexity of Model-Checking Call-by-Value Programs

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

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

Abstract. This paper studies the complexity of the reachability problem (a typical and practically important instance of the model-checking problem) for simply-typed call-by-value programs with recursion, Boolean values, and non-deterministic branch, and proves the following results. (1) The reachability problem for order-3 programs is nonelementary. Thus, unlike in the call-by-name case, the order of the input program does not serve as a good measure of the complexity. (2) Instead, the *depth* of types is an appropriate measure: the reachability problem for depth-n programs is n-EXPTIME complete. In particular, the previous upper bound given by the CPS translation is not tight. The algorithm used to prove the upper bound result is based on a novel intersection type system, which we believe is of independent interest.

1 Introduction

A promising approach to verifying higher-order functional programs is to use higher-order model checking [7, 8, 15], which is a decision problem about the trees generated by higher-order recursion schemes. Various verification problems such as the reachability problem and the resource usage verification [5] are reducible to the higher-order model checking [8].

This paper addresses a variant of the higher-order model checking, namely, the reachability problem for simply-typed *call-by-value* Boolean programs. It is the problem to decide, given a program with Boolean primitives and a special constant meaning the failure, whether the evaluation of the program fails. This is a practically important problem that can be a basis for verification of programs written in call-by-value languages such as ML and OCaml. In fact, MoCHi [11], a software model-checker for a subset of OCaml, reduces a verification problem to a reachability problem for a call-by-value Boolean program.

In the previous approach [11], the rechability problem for call-by-value programs was reduced to that for call-by-name programs via the CPS transformation. From a complexity theoretic point of view, however, this reduction via the CPS transformation has a bad effect: the order of a function is raised by 2 for each increase of the arity of the function. Since the reachability of order-n call-by-name programs is $(n-1)$-EXPTIME complete in general, the approach may suffer from double exponential blow-up of the time complexity for each increase

of the largest arity in a program. Thus, important questions are: Is the double exponential blow-up of the time complexity (with respect to the arity increase) inevitable? If not, what is the exact complexity of the reachability problem for call-by-value programs, and how can we achieve the exact complexity?

The above questions are answered in this paper. We first show that the *single* exponential blow-up with respect to the arity increase is inevitable for programs of order-3 or higher. This implies that when the arity is not fixed, the reachability problem for order-3 call-by-value programs is nonelementary. The key observation used in the proof is that the subset of natural numbers $\{0, 1, \ldots, \mathbf{exp}_n(2) - 1\}$ (here $\mathbf{exp}_n(k)$ is the nth iterated exponential function, defined by $\mathbf{exp}_0(k) = k$ and $\mathbf{exp}_{n+1}(k) = 2^{\mathbf{exp}_n(k)}$) can be embedded into the set of values of the type $\overbrace{\mathbb{B} \to \mathbb{B} \to \cdots \to \mathbb{B}}^{n} \to \mathbb{B}$ by using non-determinism.

Second, we show the *depth* of types is an appropriate measure, i.e. the reachability problem for depth-n programs is n-EXPTIME complete. The depth of function type is defined by $\mathit{depth}(\kappa \to \kappa') = \max\{\mathit{depth}(\kappa) + 1, \mathit{depth}(\kappa') + 1\}$. In particular, the previous bound given by the CPS translation is not tight. To prove the upper-bound, we develop a novel intersection type system that completely characterises programs that reach the failure. Since the target is a call-by-value language with effects (i.e. divergence, non-determinism and failure), the proposed type system is much different from that for call-by-name calculi [18, 7, 9], which we believe is of the independent interest.

Organisation of the paper Section 2 defines the problem addressed in the paper. Section 3 proves that the reachability problem for order-3 programs is nonelementary. Section 4 provides a sketch of the proof of n-EXPTIME hardness of the reachability problem for depth-n programs. In Section 5, we develop an intersection type system that characterises the reachability problem, and a type-checking algorithm. We discuss related work in Section 6 and conclude in Section 7.

2 Call-by-value Reachability Problem

The target language of the paper is a simply-typed call-by-value calculus with recursion, product types (restricted to argument positions), Boolean and non-deterministic branch. Simple types are called *sorts* in order to avoid confusion with intersection types introduced later. The sets of *sorts*, *terms* and *function definitions* (*definitions* for short) are defined by the following grammar:

$$
\begin{array}{lllll}
(\textit{Sorts}) & \kappa, \iota & ::= & \mathbb{B} \mid \kappa_1 \times \ldots \times \kappa_n \to \iota \\
(\textit{Terms}) & s, t, u & ::= & x \mid f \mid \lambda\langle x_1, \ldots, x_n \rangle.t \mid t\ \langle u_1, \ldots, u_n \rangle \\
 & & \mid & t \oplus u \mid \mathtt{t} \mid \mathtt{f} \mid \mathbf{if}(t, u_1, u_2) \mid \mathfrak{F}_\kappa \mid \Omega_\kappa \\
(\textit{Definitions}) & D & ::= & \{f_i = \lambda\langle x_{i,1}, \ldots, x_{i,n_i} \rangle.t_i\}_{i \le m},
\end{array}
$$

where $\langle x_1, \ldots, x_n \rangle$ (resp. $\langle u_1, \ldots, u_n \rangle$) is a non-empty sequence of variables (resp. terms). The sort $\mathbb{B}$ is for Boolean values and the sort $\kappa_1 \times \ldots \times \kappa_n \to \iota$ is for functions that take an n-tuple as the argument and returns a value of ι. A term

$$\frac{b \in \{\mathtt{t}, \mathtt{f}\}}{\Delta \mid \mathcal{K} \vdash b :: \mathbb{B}} \qquad \frac{x :: \kappa \in \mathcal{K}}{\Delta \mid \mathcal{K} \vdash x :: \kappa} \qquad \frac{f :: \kappa \in \Delta}{\Delta \mid \mathcal{K} \vdash f :: \kappa} \qquad \frac{\Delta \mid \mathcal{K}, \vec{x} :: \vec{\kappa} \vdash t :: \iota}{\Delta \mid \mathcal{K} \vdash \lambda\langle\vec{x}\rangle.t :: \vec{\kappa} \to \iota}$$

$$\frac{\Delta \mid \mathcal{K} \vdash t :: \vec{\kappa} \to \iota' \qquad \Delta \mid \mathcal{K} \vdash \vec{u} :: \vec{\kappa}}{\Delta \mid \mathcal{K} \vdash t\,\langle\vec{u}\rangle :: \iota} \qquad \frac{\Delta \mid \mathcal{K} \vdash t :: \mathbb{B} \quad \Delta \mid \mathcal{K} \vdash u_i :: \kappa\ (i \in \{1,2\})}{\Delta \mid \mathcal{K} \vdash \mathbf{if}(t, u_1, u_2) :: \kappa}$$

$$\frac{\Delta \mid \mathcal{K} \vdash t :: \kappa \qquad \Delta \mid \mathcal{K} \vdash u :: \kappa}{\Delta \mid \mathcal{K} \vdash t \oplus u :: \kappa} \qquad \frac{}{\Delta \mid \mathcal{K} \vdash \mathfrak{F}_\kappa :: \kappa} \qquad \frac{}{\Delta \mid \mathcal{K} \vdash \Omega_\kappa :: \kappa}$$

Fig. 1. Sorting rules for terms

is a variable x, a function symbol f (that is a variable expected to be defined in D), an abstraction $\lambda\langle x_1, \ldots, x_n\rangle.t$ that takes an n-tuple as its argument, an application $t\,\langle u_1, \ldots, u_n\rangle$ of t to n-tuple $\langle u_1, \ldots, u_n\rangle$, a non-deterministic branch $t_1 \oplus t_2$, a truth value ($\mathtt{t}$ or $\mathtt{f}$), a conditional branch $\mathbf{if}(t, u_1, u_2)$, a special constant $\mathfrak{F}_\kappa$ (standing for 'Fail') to which the reachability is considered, or divergence Ω_κ. A function definition is a finite set of elements of the form $f = \lambda\langle x_1, \ldots, x_n\rangle.t$, which defines functions by mutual recursion. If $(f = \lambda\langle\vec{x}\rangle.t) \in D$, we write $D(f) = \lambda\langle\vec{x}\rangle.t$. The domain $\mathrm{dom}(D)$ of D is $\{f \mid (f = \lambda\langle\vec{x}\rangle.t) \in D\}$.

For notational convenience, we use the following abbreviations. We write $\vec{x}$ for a *non-empty* sequence of variables $x_1, \ldots, x_n$, and simply write $\lambda\langle x_1, \ldots, x_n\rangle.t$ as $\lambda\langle\vec{x}\rangle.t$. Similarly, $t\,\langle u_1, \ldots, u_n\rangle$ is written as $t\,\langle\vec{u}\rangle$, where $\vec{u}$ indicates the sequence $u_1, \ldots, u_n$, and $\kappa_1 \times \ldots \times \kappa_n \to \iota$ as $\vec{\kappa} \to \iota$, where $\vec{\kappa} = \kappa_1, \ldots, \kappa_n$. Note that $\vec{\kappa} \to \iota$ is *not* $\kappa_1 \to \cdots \to \kappa_n \to \iota$. Sort annotation of $\mathfrak{F}_\kappa$ and Ω_κ are often omitted. For a 1-tuple $\langle t\rangle$, we often write just t.

The sort system is defined straightforwardly. A *sort environment* is a finite set of sort bindings of the form $x :: \kappa$ (here a double-colon is used for sort bindings and judgements, in order to distinguish them from type bindings and judgements). We write $\mathcal{K}(x) = \kappa$ if $x :: \kappa \in \mathcal{K}$. A *sort judgement* is of the form $\Delta \mid \mathcal{K} \vdash t :: \kappa$, where Δ is the sort environment for function symbols and $\mathcal{K}$ is the sort environment for free variables of t. Given sequences $\vec{x}$ and $\vec{\kappa}$ of the same length, we write $\vec{x} :: \vec{\kappa}$ for $x_1 :: \kappa_1, \ldots, x_n :: \kappa_n$. Given sequences $\vec{t}$ and $\vec{\kappa}$ of the same length, we write $\Delta \mid \mathcal{K} \vdash \vec{t} :: \vec{\kappa}$ just if we have $\Delta \mid \mathcal{K} \vdash t_i :: \kappa_i$ for all $i \leq n$, where n is the length of $\vec{t}$. The sorting rules are listed in Fig. 1.

When term t does not contain function symbols, we simply write $\emptyset \mid \mathcal{K} \vdash t :: \kappa$ as $\mathcal{K} \vdash t :: \kappa$. We assume that terms in the sequel are explicitly typed, i.e. every term is equipped with a sort derivation for it and we can freely refer to sorts of subterms and variables in the term. For function definitions, a judgement is of the form $\vdash D :: \Delta$, which is derived by the following rule:

$$\frac{\Delta \mid \emptyset \vdash D(f) :: \kappa \qquad (\text{for every } f :: \kappa \in \Delta)}{\vdash D :: \Delta}$$

A *program* is a pair of a definition D and a term t of the ground sort $\mathbb{B}$ with $\vdash D :: \Delta$ and $\Delta \mid \emptyset \vdash t :: \mathbb{B}$ for some Δ. A program is written as $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$. A program $\mathtt{let\ rec}\ \emptyset\ \mathtt{in}\ t$ with no function symbols is simply written as t.

The set of *values* is defined by: $v, w ::= \lambda\langle\vec{x}\rangle.t \mid \mathtt{t} \mid \mathtt{f}$. Recall that $\vec{x}$ is a non-empty sequence. *Evaluation contexts* are defined by: $E ::= \Box \mid E\ \langle\vec{t}\rangle \mid v\ \langle w_1, \ldots, w_{k-1}, E, t_{k+1}, \ldots, t_n\rangle \mid \mathbf{if}(E, t_1, t_2)$. Therefore arguments are evaluated left-to-right. The reduction relation on terms is defined by the rules below:

$$\begin{array}{lll} E[(\lambda\langle\vec{x}\rangle.t)\ \langle\vec{v}\rangle] \longrightarrow E[\,[\vec{v}/\vec{x}]t\,] & E[t_1 \oplus t_2] & \longrightarrow E[t_i] \quad (\text{for } i = 1, 2) \\ E[\mathbf{if}(\mathtt{t}, t_1, t_2)] \longrightarrow E[t_1] & E[\mathbf{if}(\mathtt{f}, t_1, t_2)] & \longrightarrow E[t_2]. \end{array}$$

We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$. The reduction relation is not deterministic because of the non-deterministic branch. A closed well-typed term t cannot be reduced just if (1) t is a value, (2) $t = E[\mathfrak{F}]$ or (3) $t = E[\Omega]$. In the second case, t immediately fails and in the third case, t never fails since Ω diverges. So we do not need to consider further reduction steps for $E[\mathfrak{F}]$ and $E[\Omega]$. By this design choice, $\longrightarrow$ is terminating.

Lemma 1. *If $\emptyset \vdash t :: \kappa$, then t has no infinite reduction sequence.*

Given a function definition D, the reduction relation $\longrightarrow_D$ is defined by the same rules as $\longrightarrow$ and the following additional rule:

$$E[f] \longrightarrow_D E[D(f)].$$

We write $\longrightarrow_D^*$ for the reflexive and transitive closure of $\longrightarrow_D$. Note that reduction by $\longrightarrow_D$ does not terminate in general.

Definition 1 (Reachability Problem). We say a program $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ *fails* if $t \longrightarrow_D^* E[\mathfrak{F}]$ for some E. The *reachability problem* is the problem to decide whether a given program fails.

Example 1. Let $t_0 = \lambda f.\mathbf{if}(f\,\mathtt{t}, \mathbf{if}(f\,\mathtt{t}, \Omega, \mathfrak{F}), \Omega)$, which calls the argument f (at most) twice with the same argument $\mathtt{t}$ and fails just if the first call returns $\mathtt{t}$ and the second call $\mathtt{f}$. Let $u_0 = (\lambda x.\mathtt{t}) \oplus (\lambda x.\mathtt{f})$ and $e_1 = t_0\,u_0$. Then e_1 has just two reduction sequences starting from $e_1 \longrightarrow t_0\,(\lambda x.\mathtt{t})$ and $e_1 \longrightarrow t_0\,(\lambda x.\mathtt{f})$, both of which do not fail. In the call-by-name setting, however, e_1 would fail since

$$\begin{array}{l} e_1 \longrightarrow \mathbf{if}(u_0\,\mathtt{t}, \mathbf{if}(u_0\,\mathtt{t}, \Omega, \mathfrak{F}), \Omega) \longrightarrow \mathbf{if}((\lambda x.\mathtt{t})\,\mathtt{t}, \mathbf{if}(u_0\,\mathtt{t}, \Omega, \mathfrak{F}), \Omega) \\ \quad \longrightarrow^* \mathbf{if}(u_0\,\mathtt{t}, \Omega, \mathfrak{F}) \longrightarrow \mathbf{if}((\lambda x.\mathtt{f})\,\mathtt{t}, \Omega, \mathfrak{F}) \longrightarrow^* \mathfrak{F}. \end{array}$$

Consider the program $e_1' = t_0\,u_0'$ where $u_0' = \lambda x.(\mathtt{t} \oplus \mathtt{f})$, in which the non-deterministic branch is delayed by the abstraction. Then e_1' would fail both in call-by-name and in call-by-value.

Example 2. Consider the program $P_2 = \mathtt{let\ rec}\ D_2\ \mathtt{in}\ e_2$, where $D_2 = \{f = \lambda x.f\,x\}$ and $e_2 = (\lambda y.\mathfrak{F})\,(f\,\mathtt{t})$. Then P_2 never fails because

$$e_2 = (\lambda y.\mathfrak{F})\,(f\,\mathtt{t}) \longrightarrow_{D_2} (\lambda y.\mathfrak{F})\,((\lambda x.f\,x)\,\mathtt{t}) \longrightarrow_{D_2} (\lambda y.\mathfrak{F})\,(f\,\mathtt{t}) = e_2 \longrightarrow_{D_2} \cdots.$$

In the call-by-name case, however, P_2 would fail since $(\lambda x.\mathfrak{F})\,(f\,\mathtt{t}) \longrightarrow \mathfrak{F}$.

Example 3. Consider the program $e_3 = (\lambda x.\mathtt{t})\,\mathfrak{F}$. Then e_3 (immediately) fails because $e_3 = E[\mathfrak{F}]$, where $E = (\lambda x.\mathtt{t})\,\Box$. In contrast, e_3 would not fail in the call-by-name setting, in which E is not an evaluation context and $e_3 \longrightarrow \mathtt{t}$.

We give a technically convenient characterisation of the reachability problem. Let $\{f_1, \ldots, f_n\}$ be the set of function symbols in D. The *mth approximation* of f_i, written F_i^m, is the term obtained by expanding the definition m times, as is formally defined below:

$$\begin{array}{ll} F_i^0 & = \lambda\langle x_1, \ldots, x_k\rangle.\Omega_\iota \qquad\qquad (\text{where } f_i :: \kappa_1 \times \ldots \times \kappa_k \to \iota \in \Delta) \\ F_i^{m+1} & = [F_1^m/f_1, \ldots, F_n^m/f_n](D(f_i)). \end{array}$$

The mth approximation of t is defined by: $[t]_D^m = [F_1^m/f_1, \ldots, F_n^m/f_n]t$.

Lemma 2. *Let $P = \mathtt{let\ rec}\ D\ \mathtt{in}\ t$ be a program. Then $t \longrightarrow_D^* E[\mathfrak{F}]$ for some E if and only if $[t]_D^n \longrightarrow^* E'[\mathfrak{F}]$ for some n and E'.*

Size of terms and programs The size of sorts is inductively defined by $|\mathbb{B}| = 1$ and $|\kappa_1 \times \ldots \times \kappa_n \to \iota| = 1 + |\iota| + \sum_{i=1}^n |\kappa_i|$. The size of sort environments is given by $|\mathcal{K}| = \sum_{x::\kappa \in \mathcal{K}} |\kappa|$. The size of a term is defined straightforwardly (e.g. $|x| = 1$ and $|t\,\langle u_1, \ldots, u_n\rangle| = 1 + |t| + \sum_{i=1}^n |u_i|$) except for the abstraction $|\lambda\langle x_1, \ldots, x_n\rangle.t| = 1 + |t| + \sum_{i=1}^n (1 + |\kappa_i|)$, where κ_i is the sort of x_i. Here a term t is considered to be explicitly sorted, and thus the size of annotated sorts should be added. For programs, $|\mathtt{let\ rec}\ D\ \mathtt{in}\ t| = |t| + \sum_{f \in \mathrm{dom}(D)} |D(f)|$.

Order and depth of programs Order is a well-known measure that characterises complexity of the *call-by-name* reachability problem [10, 15] (it is $(n-1)$-EXPTIME complete for order-n programs) and, as we shall see, depth characterises complexity in the call-by-value case. *Order* and *depth* of sorts are defined by:

$$\begin{array}{l} \mathit{order}(\mathbb{B}) = \mathit{depth}(\mathbb{B}) = 0 \\ \mathit{order}(\vec{\kappa} \to \iota) = \max\{\mathit{order}(\iota), \quad \mathit{order}(\kappa_1)+1, \ldots, \mathit{order}(\kappa_n)+1\} \\ \mathit{depth}(\vec{\kappa} \to \iota) = \max\{\mathit{depth}(\iota)+1, \mathit{depth}(\kappa_1)+1, \ldots, \mathit{depth}(\kappa_n)+1\} \end{array}$$

For a sort environment, $\mathit{depth}(\mathcal{K}) = \max\{\mathit{depth}(\kappa) \mid x :: \kappa \in \mathcal{K}\}$. Order and depth of judgements are defined by $\varphi(\Delta \mid \mathcal{K} \vdash t :: \kappa) = \varphi(\kappa)$, where $\varphi \in \{\mathit{order}, \mathit{depth}\}$. The order of a sort derivation is the maximal order of judgements in the derivation. The order of a sorted term t is the order of its sort derivation $\Delta \mid \mathcal{K} \vdash t :: \kappa$. The order of a program $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ is the maximal order of terms t and $D(f)$ ($f \in \mathrm{dom}(D)$). The depth of derivations, sorted terms and programs are defined similarly.

3 Order-3 Reachability is Nonelementary

This section proves the following theorem.

Theorem 1. *The reachability problem for order-3 programs is nonelementary.*

The key observation is that, for every n, the subset of natural numbers $\{0, 1, \ldots, \mathbf{exp}_n(2) - 1\}$ can be implemented by $\overbrace{\mathbb{B} \to \cdots \to \mathbb{B}}^{n} \to \mathbb{B}$ in a certain sense (see Definition 2). The non-determinism of the calculus is essential to the construction. Note that in the call-by-name case, the set of closed terms (modulo observational equivalence) of this sort can be bounded by 4^{4^n}, since

$$\overbrace{\mathbb{B} \to \cdots \to \mathbb{B}}^{n} \to \mathbb{B} \cong \overbrace{\mathbb{B} \times \cdots \times \mathbb{B}}^{n} \to \mathbb{B}.$$

The proof in this section can be sketched as follows. Let $L \subseteq \{0,1\}^*$ be a language in n-EXPSPACE. We can assume without loss of generality that there exists a Turing machine M that accepts L and runs in space $\mathbf{exp}_n(x)$ (here x is the size of the input). Given a word w, we reduce its acceptance by M to the reachability problem of a program (say $P_{M,w}$) of the call-by-value calculus in Section 2 extended to have natural numbers up to $N \geq \mathbf{exp}_n(x)$ (Lemma 3). The order of $P_{M,w}$ is independent from M and w: it is 3 when the order of the natural number type is defined to be 1. Recall that the natural numbers up to $\mathbf{exp}_{n+x}(2) \geq \mathbf{exp}_n(x)$ can be implemented by the order-1 sort $\overbrace{\mathbb{B} \to \cdots \to \mathbb{B}}^{n+x} \to \mathbb{B}$. By replacing natural numbers in $P_{M,w}$ with the implementation, the acceptance of w by M can be reduced to the reachability problem of an order-3 program without natural numbers.

3.1 Simulating Turing Machine by Program with Natural Numbers

First of all, we define programs with natural numbers up to N, which is an extension of the typed calculus presented in Section 2. The syntax of sorts and terms is given by:

$$\begin{array}{lll} (\textit{Sorts}) & \kappa, \iota & ::= \cdots \mid \mathbb{N} \\ (\textit{Terms}) & s, t, u & ::= \cdots \mid \mathtt{S} \mid \mathtt{P} \mid \mathtt{EQ} \mid \underline{0} \mid \underline{1} \mid \cdots \mid \underline{N-1} \end{array}$$

The extended calculus has an additional ground sort $\mathbb{N}$ for (bounded) natural numbers. Constants $\mathtt{S}$ and $\mathtt{P}$ are functions of sort $\mathbb{N} \to \mathbb{N}$ meaning the successor and the predecessor functions, respectively, and $\mathtt{EQ}$ is a constant of sort $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$ which checks if two arguments are equivalent. A constant $\underline{n}$ indicates the natural number n. The set of values is defined by: $v ::= \cdots \mid \mathtt{S} \mid \mathtt{P} \mid \mathtt{EQ} \mid \underline{n}$. Function definitions and evaluation contexts are given by the same syntax as in Section 2, but terms and values may contain natural numbers. The additional reduction rules are given by

$$\begin{array}{rll} E[\mathtt{S}\,\underline{n}] & \longrightarrow_D E[\underline{n+1}] & (\text{if } n+1 < N) \\ E[\mathtt{P}\,\underline{n}] & \longrightarrow_D E[\underline{n-1}] & (\text{if } n-1 \geq 0) \\ E[\mathtt{EQ}\,\langle \underline{n}, \underline{n} \rangle] & \longrightarrow_D E[\mathtt{t}] & \\ E[\mathtt{EQ}\,\langle \underline{n}, \underline{m} \rangle] & \longrightarrow_D E[\mathtt{f}] & (\text{if } n \neq m). \end{array}$$

Note that $E[\mathtt{S}\,\underline{N-1}]$ and $E[\mathtt{P}\,\underline{0}]$ get stuck. A *program with natural numbers up to* N is a pair of a function definition D and a term t of sort $\mathbb{B}$, written as $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$. We assume that programs in the sequel do not contain constant numbers except for $\underline{0}$. The order of $\mathbb{N}$ is defined as 1.

Lemma 3. *Let $L \subseteq \{0,1\}^*$ be a language and M be a deterministic Turing machine accepting L that runs in space $\mathbf{exp}_n(x)$ for some n. Then, for every word $w \in \{0,1\}^*$ of length k and natural number $N \geq \mathbf{exp}_n(k)$, one can construct a program $P_{M,w}$ with natural numbers up to N such that $P_{M,w}$ fails if and only if $w \in L$. Furthermore $P_{M,w}$ is of order-3 and can be constructed in polynomial time with respect to k.*

Proof. Let M be a Turing machine with states Q and tape symbols Σ and w be a word of length k. We can assume without loss of generality that $Q = \{\mathtt{t},\mathtt{f}\}^q$ (that is, the set of all sequences of length q consisting of $\mathtt{t}$ and $\mathtt{f}$) and $\Sigma = \{\mathtt{t},\mathtt{f}\}^l$.

A configuration is expressed as a value of sort[4]

$$\mathit{Config} = \overbrace{\mathbb{B} \times \cdots \times \mathbb{B}}^{q} \times \overbrace{(\mathbb{N} \to \mathbb{B}) \times \cdots \times (\mathbb{N} \to \mathbb{B})}^{l} \times \mathbb{N},$$

where the first part represents the current state, the second part the tape and the third part the position of the tape head. The program $P_{M,w}$ has one recursive function *isAccepted* of sort $\mathit{Config} \to \mathbb{B}$. It checks if the current state is a final state and it fails if so. Otherwise it computes the next configuration and passes it to *isAccepted* itself. The body of the program generates the initial configuration determined by w and passes it to the function *isAccepted*.

Clearly we can construct $P_{M,w}$ in polynomial time with respect to k (the length of w) and the order of $P_{M,w}$ is 3. □

3.2 Implementing Natural Numbers

Let ν_n be the order-1 sort defined by $\nu_0 = \mathbb{B}$ and $\nu_{n+1} = \mathbb{B} \to \nu_n$. We shall show that natural numbers up to $\mathbf{exp}_n(2)$ can be implemented as values of ν_n.

Intuitive Explanation We explain the intuition behind the construction by using the set-theoretic model. Let $\mathbf{N} = \{0,1,\ldots,N-1\}$. We explain the way to express the set $2^{\mathbf{N}} \cong \{0,1,\ldots,2^N-1\}$ as (a subset of) non-deterministic functions of $\mathbb{B} \to \mathbf{N}$, i.e. functions of $\mathbb{B} \to \mathcal{P}(\mathbf{N})$, where $\mathcal{P}(\mathbf{N})$ is the powerset of $\mathbf{N}$. The set $(\mathbb{B} \Rightarrow \mathbf{N}) \subseteq (\mathbb{B} \to \mathcal{P}(\mathbf{N}))$ is defined by:

$$(\mathbb{B} \Rightarrow \mathbf{N}) = \{f : \mathbb{B} \to \mathcal{P}(\mathbf{N}) \mid f(\mathtt{t}) \cup f(\mathtt{f}) = \mathbf{N} \text{ and } f(\mathtt{t}) \cap f(\mathtt{f}) = \emptyset\}.$$

[4] Strictly speaking, it is not a sort in our syntax because products are restricted to argument positions. But there is no problem since occurrences of *Config* in the following construction are also restricted to argument positions.

In other words, $f \in \mathbb{B} \to \mathcal{P}(\mathbf{N})$ is in $\mathbb{B} \Rightarrow \mathbf{N}$ if and only if, for every $i \in \mathbf{N}$, exactly one of $i \in f(\mathtt{t})$ and $i \in f(\mathtt{f})$ holds. Hence a function $f : \mathbb{B} \Rightarrow \mathbf{N}$ determines a function of $\mathbf{N} \to \mathbb{B}$, say $\hat{f}$, defined by $\hat{f}(i) = b$ iff $i \in f(b)$ $(b \in \{\mathtt{t}, \mathtt{f}\})$.

There is a bijection between the set of functions $\mathbf{N} \to \mathbb{B}$ and the subset of natural numbers $\{0, 1, \ldots, 2^N - 1\}$, given by binary encoding, i.e. $(\hat{f} : \mathbf{N} \to \mathbb{B}) \mapsto \sum_{i<N, \hat{f}(i)=\mathtt{t}} 2^i$. For example, consider the case that $N = 4$ and $\mathbf{N} = \{0, 1, 2, 3\}$. Then 6 (= 0110 in binary) is represented by $\hat{f}_6$ such that $\hat{f}_6(0) = \hat{f}_6(3) = \mathtt{f}$ and $\hat{f}_6(1) = \hat{f}_6(2) = \mathtt{t}$. Therefore f_6 is given by $f_6(\mathtt{t}) = \{1, 2\}$ and $f_6(\mathtt{f}) = \{0, 3\}$.

Now let us consider the way to define operations such as the successor, predecessor and equality test. The key fact is that there is a term (say $\mathtt{get}$) that computes $\hat{f}(i)$ for $f \in \mathbb{B} \Rightarrow \mathbf{N}$ and $i \in \mathbf{N}$, and there exists a term (say $\mathtt{put}$) that computes $g \in \mathbb{B} \Rightarrow \mathbf{N}$ such that $\hat{g} = \hat{f}[i \mapsto b]$ for $f \in \mathbb{B} \Rightarrow \mathbf{N}$, $i \in \mathbf{N}$ and $b \in \{\mathtt{t}, \mathtt{f}\}$. They are given by the following informal equations:

$$\begin{aligned} \mathtt{get}\,\langle f, i\rangle &= \mathbf{if}(f\,\mathtt{t} = i,\ \mathtt{t},\ \Omega) \ \oplus\ \mathbf{if}(f\,\mathtt{f} = i,\ \mathtt{f},\ \Omega) \\ \mathtt{put}\,\langle f, i, b\rangle &= \lambda c^{\mathbb{B}}.\big(\mathbf{if}(b = c, i, \Omega) \ \oplus\ ((\lambda j.\mathbf{if}(i \neq j, j, \Omega))\,(f\,c))\big) \end{aligned}$$

where $f :: \mathbb{B} \to \mathbf{N}$ and $i, j :: \mathbf{N}$ and $b, c :: \mathbb{B}$. Note that $\mathtt{put}$ would be incorrect in the call-by-name setting. By using these functions, we can write operations like successor, predecessor and equality test for $\mathbb{B} \Rightarrow \mathbf{N}$. For example, the equality test eq can be defined by $eq = \lambda\langle f, g\rangle.e\,\langle f, g, N-1\rangle$, where e is given by the following recursive definition:

$$e\,\langle f, g, i\rangle = \mathbf{if}((\mathtt{get}\,\langle f, i\rangle) = (\mathtt{get}\,\langle g, i\rangle)), \quad \mathbf{if}(i = 0,\ \mathtt{t},\ e\,\langle f, g, (i-1)\rangle), \quad \mathtt{f}).$$

Formal Development We formally define the notion of implementations and show that replacement of natural numbers with its implementations preserves reachability.

Definition 2 (Implementation of Natural Numbers). Let $\mathbf{N}$ be the tuple $(N, D, \kappa, \{V_i\}_{i\in\{0,1,\ldots,N-1\}}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$, where N is a natural number, D is a function definition, κ is a sort, $\{V_i\}_i$ is an indexed set of pairwise disjoint sets of closed values of sort κ, $\mathbf{eq}$ is a closed value of sort $\kappa \times \kappa \to \mathbb{B}$, $\mathbf{s}$ and $\mathbf{p}$ are closed values of sort $\kappa \to \kappa$, and $\mathbf{z}$ and $\mathbf{max}$ are closed values of sort κ. Here we consider terms *without* natural numbers. We say $\mathbf{N}$ is an *implementation of natural numbers up to* N just if the following conditions hold (here $V = \bigcup_i V_i$).

- For every $v, v' \in V$, evaluation of $\mathbf{eq}\,\langle v, v'\rangle$, $\mathbf{s}\,v$ and $\mathbf{p}\,v$ under D never fails.
- $\mathbf{z} \in V_0$ and $\mathbf{max} \in V_{N-1}$.
- For every $v \in V_n$ and $v' \in V_{n'}$, $\mathbf{eq}\,\langle v, v'\rangle \longrightarrow_D^* \mathtt{t}$ if and only if $n = n'$, and $\mathbf{eq}\,\langle v, v'\rangle \longrightarrow_D^* \mathtt{f}$ if and only if $n \neq n'$.
- For every $v \in V_n$, $\mathbf{s}\,v \longrightarrow_D^* v'$ implies $v' \in V_{n+1}$ and if $n + 1 < N$ then $\mathbf{s}\,v \longrightarrow_D^* v'$ for some $v' \in V_{n+1}$. Similarly, $\mathbf{p}\,v \longrightarrow_D^* v'$ implies $v' \in V_{n-1}$ and if $n \geq 1$ then $\mathbf{p}\,v \longrightarrow_D^* v'$ for some $v' \in V_{n-1}$. (Here we define $V_{-1} = V_{N+1} = \emptyset$.)

The sort of $\mathbf{N}$ is κ and the order of $\mathbf{N}$ is that of κ.

Given an implementation $\mathbf{N}$ of natural numbers up to N and a term t with natural numbers up to N, we write $t^{\mathbf{N}}$ for the term without natural numbers obtained by replacing constants with values given by $\mathbf{N}$, e.g.,

$$\underline{0}^{\mathbf{N}} = \mathbf{z} \qquad \mathtt{S}^{\mathbf{N}} = \mathbf{s} \qquad (t\,u)^{\mathbf{N}} = t^{\mathbf{N}}\,u^{\mathbf{N}} \qquad (\lambda x.t)^{\mathbf{N}} = \lambda x.(t^{\mathbf{N}}).$$

Note that programs do not contain constant numbers except for $\underline{0}$ by definition. Given a function definition D, $D^{\mathbf{N}}$ can be defined straightforwardly. See Appendix C.1 for the concrete definition.

Lemma 4. *Let* $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ *be a program with natural numbers up to* N*, and* $\mathbf{N}$ *be an implementation of natural numbers up to* N*. Then* $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ *fails if and only if* $\mathtt{let\ rec}\ D^{\mathbf{N}}\ \mathtt{in}\ t^{\mathbf{N}}$ *fails.*

Given a natural number $n \geq 1$, we present an implementation of natural numbers up to $\mathbf{exp}_n(2)$ whose order is 1. By using the implementation to the program constructed in Lemma 3, the nonelementary result for the reachability problem for order-3 programs is established.

For every n, we shall define an implementation $\mathbf{N}(n)$ of natural numbers up to $\mathbf{exp}_n(2)$ by induction on n. As for the base case, the natural numbers up to $\mathbf{exp}_0(2) = 2$ (i.e. $\{0, 1\}$) can be naturally implemented by using $\mathbb{B}$. We call this implementation $\mathbf{N}(0)$. As for the induction step, assuming an implementation $\mathbf{N} = (N, D, \kappa, \{V_i\}_i, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$ of natural numbers up to N, it suffices to construct an implementation of natural numbers up to 2^N, say ${}^{\mathbb{B}}\mathbf{N} = (2^N, D \cup D', \mathbb{B} \to \kappa, \{V'_i\}_{i \in \{0,1,\ldots,2^N-1\}}, \mathbf{eq}', \mathbf{s}', \mathbf{p}', \mathbf{z}', \mathbf{max}')$.

- The additional function definition D' defines `get`, `put` and other auxiliary functions used to define $\mathbf{s}'$ and others. The definitions of `get` and `put` are:

$$\mathtt{get} = \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}\rangle.\quad \mathbf{if}(\mathbf{eq}\,\langle x\,\mathtt{t}, i\rangle,\ \mathtt{t},\ \Omega)\ \oplus\ \mathbf{if}(\mathbf{eq}\,\langle x\,\mathtt{f}, i\rangle,\ \mathtt{f},\ \Omega)$$

$$\mathtt{put} = \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}, b^{\mathbb{B}}\rangle.\lambda c^{\mathbb{B}}.\ \big(\mathbf{if}(b = c, i, \Omega)\ \oplus\ ((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle i, j\rangle, \Omega, j))\,(x\,c))\big)$$

- Let $m < 2^N$ and $b_{N-1} \ldots b_0$ be its binary representation. Then V'_m is the set of values v of sort $\mathbb{B} \to \kappa$ such that
  1. $b_i = 1$ iff $v\,\mathtt{t} \longrightarrow^* v'$ for some $v' \in V_i$,
  2. $b_i = 0$ iff $v\,\mathtt{f} \longrightarrow^* v'$ for some $v' \in V_i$, and
  3. $v\,\mathtt{t} \longrightarrow^* v'$ or $v\,\mathtt{f} \longrightarrow^* v'$ implies $v' \in \bigcup_{i \in \{0,\ldots,N-1\}} V_i$.

Here $x = y$ is the shorthand for $\mathbf{if}(x, \mathbf{if}(y, \mathtt{t}, \mathtt{f}), \mathbf{if}(y, \mathtt{f}, \mathtt{t}))$. For $n \geq 1$, we define $\mathbf{N}(n+1) = {}^{\mathbb{B}}(\mathbf{N}(n))$. See Appendix C.3 for the concrete definition of ${}^{\mathbb{B}}\mathbf{N}$ and the proof of the following lemma.

Lemma 5. $\mathbf{N}(n)$ *is an implementation of natural numbers up to* $\mathbf{exp}_n(2)$*. Furthermore, the sort, the function definition and the operations of* $\mathbf{N}(n)$ *can be constructed in time polynomial with respect to* n*.*

Proof (Theorem 1). The claim follows from Lemmas 3, 4 and 5. Note that (i) $\mathbf{exp}_n(x) \leq \mathbf{exp}_{n+x}(2)$, and (ii) given an order-n program with natural numbers up to $\mathbf{exp}_m(2)$, the replacement of natural number constants with $\mathbf{N}(m)$ can be done in time polynomial with respect to m and the size of the program, and the resulting program is of order n (provided that $n \geq 2$). □

4 Depth-n Reachability is n-EXPTIME Hard

In this section, we show a sketch of the proof of Theorem 2 below.

Theorem 2. *For every $n > 0$, the reachability problem for depth-n programs is n-EXPTIME hard.*

We reduce the emptiness problem of order-n alternating pushdown systems, which is known to be n-EXPTIME complete [4], to the reachability problem for depth-n programs. The basic idea originates from the work of Knapik et al. [6], which simulates a deterministic higher-order pushdown automaton by a safe higher-order grammar.

Since Knapik et al. [6] considered call-by-name grammars, we need to fill the gap between call-by-name and call-by-value. A problem arises when a divergent term that would not be evaluated in the call-by-name strategy appears in an argument position. We use the non-deterministic branch and the Boolean values to overcome the problem. Basically, by our reduction, every term of the ground sort is of the form $\mathtt{f} \oplus s$, and thus one can choose whether s is evaluated or not, by selecting one of the two possible reduction $\mathtt{f} \oplus s \longrightarrow \mathtt{f}$ and $\mathtt{f} \oplus s \longrightarrow s$. See Appendix D for more details.

5 Intersection-Type-Based Model-Checking Algorithm

We develop an intersection type system that completely characterises the reachability problem and give an upper bound of complexity of the reachability problem by solving the typability problem.

5.1 Types

The pre-types are given by the following grammar:

$$\begin{array}{lrcl} \textit{(Value Pre-types)} & \theta & ::= & \mathtt{t} \mid \mathtt{f} \mid \bigwedge_{i\in I}(\theta_{1,i}\times\ldots\times\theta_{n,i}\to\tau_i) \\ \textit{(Term Pre-types)} & \tau,\sigma & ::= & \theta \mid \mathfrak{F}_\kappa \end{array}$$

The index I of the intersection is a finite set. We allow I to be the empty set, and we also write $\bigwedge\emptyset$ for the type. The subscript κ of $\mathfrak{F}_\kappa$ is often omitted. We use infix notation for intersection, e.g. $(\theta_1 \to \tau_1) \wedge (\theta_2 \to \tau_2)$. The intersection connective is assumed to be associative, commutative and idempotent. Thus types $\bigwedge_{i\in I}(\theta_{1,i}\times\ldots\times\theta_{n,i}\to\tau_i)$ and $\bigwedge_{j\in J}(\theta'_{1,j}\times\ldots\times\theta'_{n,j}\to\tau'_{n,j})$ are equivalent if $\{(\theta_{1,i},\ldots,\theta_{n,i},\tau_i) \mid i\in I\}$ and $\{(\theta'_{1,j},\ldots,\theta'_{n,j},\tau'_j) \mid j\in J\}$ are equivalent sets.

Value pre-types are types for values and term pre-types are those for terms.

The value pre-type $\mathtt{t}$ is for the Boolean value $\mathtt{t}$ and $\mathtt{f}$ for the Boolean value $\mathtt{f}$. The last one is for abstractions. It can be understood as the intersection of function types of the form $\theta_1\times\ldots\times\theta_n\to\tau$. The judgement $\lambda\langle\vec{x}\rangle.t : \theta_1\times\ldots\times\theta_n\to\tau$ means that, for all values $v_i : \theta_i$ (for every $i \le n$), one has $[\vec{v}/\vec{x}]t : \tau$. For example, $\lambda x.x : \mathtt{t}\to\mathtt{t}$ and $\lambda x.x : \mathtt{f}\to\mathtt{f}$. The judgement $\lambda\langle\vec{x}\rangle.t : \bigwedge_{i\in I}(\theta_{1,i}\times\ldots\times\theta_{n,i}\to\tau_i)$

means that, for every $i \in I$, one has $\lambda\langle\vec{x}\rangle.t : \theta_{1,i} \times \ldots \times \theta_{n,i} \to \tau_i$. Therefore, $\lambda x.x : (\mathtt{t} \to \mathtt{t}) \wedge (\mathtt{f} \to \mathtt{f})$.

The term pre-type $\mathfrak{F}$ means failure, i.e. $t : \mathfrak{F}$ just if $t \longrightarrow^* E[\mathfrak{F}]$. The term pre-type θ is for terms that is reducible to a value of type θ, i.e. $t : \theta$ just if $t \longrightarrow^* v$ and $v : \theta$ for some v. For example, consider $u_0 = (\lambda x.\mathtt{t}) \oplus (\lambda x.\mathtt{f})$ and $u_0' = \lambda x.(\mathtt{t} \oplus \mathtt{f})$ in Example 1. Then $u_0 : \mathtt{t} \to \mathtt{t}$ since $u_0 \longrightarrow \lambda x.\mathtt{t}$, and $u_0 : \mathtt{t} \to \mathtt{f}$ since $u_0 \longrightarrow \lambda x.\mathtt{f}$. It is worth noting that $t : \theta_1$ and $t : \theta_2$ does not imply $t : \theta_1 \wedge \theta_2$, e.g. u_0 does not have type $(\mathtt{t} \to \mathtt{t}) \wedge (\mathtt{t} \to \mathtt{f})$. In contrast, $u_0' : (\mathtt{t} \to \mathtt{t}) \wedge (\mathtt{t} \to \mathtt{f})$. So the difference between u_0 and u_0' is captured by types.

Given a sort κ, the relation $\tau :: \kappa$, read "τ is a refinement of κ," is inductively defined by the following rules:

$$\frac{}{\mathtt{t} :: \mathbb{B}} \qquad \frac{}{\mathtt{f} :: \mathbb{B}} \qquad \frac{}{\mathfrak{F}_\kappa :: \kappa} \qquad \frac{\theta_{k,i} :: \kappa_k \qquad \tau_i :: \iota \quad (\text{for all } i \in I,\, k \in \{1,\ldots,n\})}{\bigwedge_{i\in I}(\theta_{1,i} \times \ldots \times \theta_{n,i} \to \tau_i) :: \kappa_1 \times \ldots \times \kappa_n \to \iota}$$

Note that intersection is allowed only for pre-types of the same sort. So a pre-type like $((\mathtt{t} \to \mathtt{t}) \to \mathtt{t}) \wedge (\mathtt{t} \to \mathtt{t})$ is not a refinement of any sort. A *type* is a value pre-type with its sort $\theta :: \kappa$ or a term pre-type with its sort $\tau :: \kappa$. A type is often simply written as θ or τ.

Let $\theta, \theta' :: \kappa$ be value types of the same sort. We define $\theta \wedge \theta'$ by:

$$\mathtt{t} \wedge \mathtt{t} = \mathtt{t} \qquad \mathtt{f} \wedge \mathtt{f} = \mathtt{f} \qquad (\bigwedge_{i\in I}(\vec{\theta_i} \to \tau_i)) \wedge (\bigwedge_{j\in J}(\vec{\theta_j} \to \tau_j)) = \bigwedge_{i\in I\cup J}(\vec{\theta_i} \to \tau_i)$$

and $\mathtt{t} \wedge \mathtt{f}$ and $\mathtt{f} \wedge \mathtt{t}$ are undefined.

5.2 Typing Rules

A *type environment* Γ is a finite set of type bindings of the form $x : \theta$ (here x is a variable or a function symbol). We write $\Gamma(x) = \theta$ if $x : \theta \in \Gamma$. We assume type bindings respect sorts, i.e. $x :: \kappa$ implies $\Gamma(x) :: \kappa$. A *type judgement* is of the form $\Gamma \vdash t : \tau$. The judgement intuitively means that, if each free variable x in t is bound to a value of type $\Gamma(x)$, then at least one possible evaluation of t results in a value of type τ. We abbreviate a sequence of judgements $\Gamma \vdash t_1 : \tau_1, \ldots, \Gamma \vdash t_n : \tau_n$ as $\Gamma \vdash \vec{t} : \vec{\tau}$. The typing rules are listed in Fig. 2.

Here are some notes on typing rules. Rule (Abs) can be understood as the (standard) abstraction rule followed by the intersection introduction rule. Rule (App) can be understood as the intersection elimination rule followed by the (standard) application rule. Note that intersection is introduced by (Abs) rule and eliminated by (App) rule, which is the converse of the call-by-name case [7]. Rule (Var) is designed for ensuring weakening. Rule (App-$\mathfrak{F}$1) reflects the fact that, if $t \longrightarrow^* E[\mathfrak{F}]$, then $t\,\langle\vec{u}\rangle \longrightarrow^* E'[\mathfrak{F}]$ where $E' = E\,\langle\vec{u}\rangle$. Rule (App-$\mathfrak{F}$2) reflects the fact that, if $t \longrightarrow v_0$ and $u_i \longrightarrow^* v_i$ for $i < l$, then $t\,\langle u_1, \ldots, u_{l-1}, u_l, u_{l+1}, \ldots, u_n\rangle \longrightarrow^* v_0\,\langle v_1, \ldots, v_{l-1}, E[\mathfrak{F}], u_{l+1}, \ldots, u_n\rangle$. The premises $t : \theta_0$ and $u_i : \theta_i$ $(i < l)$ ensure may-convergence of their evaluation.

Typability of a program is defined by using the notion of the nth approximation (see Section 2 for the definition). Let $P = \mathtt{let\ rec}\ D\ \mathtt{in}\ t$ be a program.

$$\frac{x : \theta \wedge \theta' \in \Gamma \text{ for some } \theta'}{\Gamma \vdash x : \theta} \quad (\textsc{Var})$$

$$\frac{b \in \{\mathtt{t}, \mathtt{f}\}}{\Gamma \vdash b : b} \quad (\textsc{Bool})$$

$$\frac{}{\Gamma \vdash \mathfrak{F} : \mathfrak{F}} \quad (\mathfrak{F})$$

$$\frac{\Gamma, \vec{x} : \vec{\theta_i} \vdash t : \tau_i \text{ for all } i \in I}{\Gamma \vdash \lambda\langle\vec{x}\rangle.t : \bigwedge_{i \in I}(\vec{\theta_i} \to \tau_i)} \quad (\textsc{Abs})$$

$$\frac{\begin{array}{l}\Gamma \vdash t : \bigwedge_{i \in I}(\vec{\theta_i} \to \tau_i) \\ \Gamma \vdash \vec{u} : \vec{\theta_l} \qquad l \in I\end{array}}{\Gamma \vdash t\,\langle\vec{u}\rangle : \tau_l} \quad (\textsc{App})$$

$$\frac{\Gamma \vdash t : \mathfrak{F}}{\Gamma \vdash t\,\langle\vec{u}\rangle : \mathfrak{F}} \quad (\textsc{App-}\mathfrak{F}1)$$

$$\frac{\begin{array}{l}\Gamma \vdash t : \theta_0 \\ \Gamma \vdash u_1 : \theta_1 \\ \quad\vdots \\ \Gamma \vdash u_{l-1} : \theta_{l-1} \\ \Gamma \vdash u_l : \mathfrak{F}\end{array}}{\Gamma \vdash t\,\langle\vec{u}\rangle : \mathfrak{F}} \quad (\textsc{App-}\mathfrak{F}2)$$

$$\frac{\Gamma \vdash t : \mathtt{t} \qquad \Gamma \vdash s_1 : \tau}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \tau} \quad (\text{C-T})$$

$$\frac{\Gamma \vdash t : \mathtt{f} \qquad \Gamma \vdash s_2 : \tau}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \tau} \quad (\text{C-F})$$

$$\frac{\Gamma \vdash t : \mathfrak{F}}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \mathfrak{F}} \quad (\text{C-}\mathfrak{F})$$

$$\frac{\exists i \in \{1, 2\} \quad \Gamma \vdash t_i : \tau}{\Gamma \vdash t_1 \oplus t_2 : \tau} \quad (\textsc{Br})$$

Fig. 2. Typing Rules

Thus t is a term of sort $\mathbb{B}$ with free occurrences of function symbols. We say the program P has type τ (written as $\vdash P : \tau$) just if $\vdash [t]^n_D : \tau$ for some n.

Soundness and completeness of the type system can be proved by using a standard technique for intersection type systems, except that Substitution and De-Substitution Lemmas are restricted to substitution of values and Subject Reduction and Expansion properties are restricted to call-by-value reductions. For more details, see Appendix E.

Theorem 3. $\vdash P : \mathfrak{F}$ *if and only if* P *fails.*

5.3 Type-Checking Algorithm and Upper Bound of Complexity

We provide an algorithm that decides the typability of a given depth-n program P in time $O(\mathbf{exp}_n(poly(|P|)))$ for some polynomial *poly*. Let $P = \mathtt{let\ rec}\ D\ \mathtt{in}\ t$ and suppose that $\vdash D :: \Delta$, $\Delta \vdash t :: \mathbb{B}$ and $\Delta = \{f_i :: \delta_i \mid i \in I\}$.

We define $\mathcal{T}(\kappa) = \{\tau \mid \tau :: \kappa\}$ and $\mathcal{T}(\Delta) = \{\Gamma \mid \Gamma :: \Delta\}$. For $\tau, \sigma \in \mathcal{T}(\kappa)$, we write $\tau \preceq \sigma$ just if $\tau = \sigma \wedge \sigma'$ for some σ'. The ordering for type environments is defined similarly. Let $\mathcal{F}_D$ be a function on $\mathcal{T}(\Delta)$, defined by:

$$\mathcal{F}_D(\Theta) = \Big\{ f : \bigwedge\{\vec{\theta} \to \tau \mid \Theta, \vec{x} : \vec{\theta} \vdash t : \tau\} \;\Big|\; (f = \lambda\langle\vec{x}\rangle.t) \in D \Big\}.$$

The algorithm to decide whether $\vdash \mathtt{let\ rec}\ D\ \mathtt{in}\ t : \mathfrak{F}$ is shown in Fig. 3.

```
1   :  Θ_0 := {f : ⋀∅ | f ∈ dom(Δ)},  Θ_1 = F_D(Θ_0),  i := 1
2   :  while Θ_i ≠ Θ_{i−1} do
2-1 :      Θ_{i+1} := F_D(Θ_i)
2-2 :      i := i + 1
3   :  if Θ_i ⊢ t : 𝔉 then yes else no
```

Fig. 3. Algorithm checking if $\vdash \texttt{let rec}\ D\ \texttt{in}\ t : \mathfrak{F}$

Termination of the algorithm comes from monotonicity of $\mathcal{F}_D$ and finiteness of $\mathcal{T}(\Delta)$. Correctness is a consequence of the following lemma and the monotonicity of the approximation (i.e. if $[t]^m_D$ fails and $m \leq m'$, then $[t]^{m'}_D$ fails).

Lemma 6. *Suppose $\Delta \mid \mathcal{K} \vdash t :: \mathbb{B}$. Then $\emptyset \vdash [t]^n_D : \tau$ if and only if $\Theta_n \vdash t : \tau$.*

We shall analyse the cost of the algorithm. For a set A, we write $\#A$ for the number of elements. The *height* of a poset A is the maximum length of strictly increasing chains in A.

Lemma 7. *Let κ be a sort of depth n. Then $\#\mathcal{T}(\kappa) \leq \mathbf{exp}_{n+1}(2|\kappa|)$ and the height of $\mathcal{T}(\kappa)$ is bounded by $\mathbf{exp}_n(2|\kappa|)$.*

Lemma 8. *Let $\Delta \mid \mathcal{K} \vdash t :: \kappa$ be a sorted term of depth n, and $\Theta :: \Delta$. Assume that depth$(\mathcal{K}) \leq n-1$. Then $A_{\Theta,t} = \{(\Gamma,\tau) \in \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa) \mid \Theta, \Gamma \vdash t : \tau\}$ can be computed in time $O(\mathbf{exp}_n(poly(|t|)))$ for some polynomial poly.*

Proof. We can compute $A_{\Theta,t}$ by induction on t. An important case is that the sort κ is of depth n. In this case, there exists $B_{\Theta,t} \subseteq \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa)$ such that (1) $(\Gamma,\tau) \in A_{\Theta,t}$ if and only if $(\Gamma,\tau') \in B_{\Theta,t}$ for some $\tau' \preceq \tau$ and (2) for each Γ, the number of elements in $B_{\Theta,t} \restriction \Gamma = \{\tau \mid (\Gamma,\tau) \in B_{\Theta,t}\}$ is bounded by $|t|$. This claim can be proved by induction on t. See Appendix H. By using $B_{\Theta,t}$ as the representation of $A_{\Theta,t}$, $A_{\Theta,t}$ can be computed in the desired bound. For other cases, one can enumerate all the elements in $A_{\Theta,t}$, since $\#A_{\Theta,t} \leq \mathbf{exp}_n(2(|\mathcal{K}| + |\kappa|)) \leq \mathbf{exp}_n(2|t|)$ (here we assume w.l.o.g. that each variable in dom$(\mathcal{K})$ appears in t). □

Theorem 4. *The reachability problem for depth-n programs is in n-EXPTIME.*

Proof. By Lemma 8, each iteration of loop 2 in Fig. 3 runs in n-EXPTIME. Since the height of $\mathcal{T}(\Delta)$ is bounded by $\mathbf{exp}_n(2|\Delta|)$, one needs at most $\mathbf{exp}_n(2|\Delta|)$ iterations for loop 2, and thus loop 2 runs in n-EXPTIME. Again by Lemma 8, step 3 can be computed in n-EXPTIME. Thus the algorithm in Fig. 3 runs in n-EXPTIME for depth-n programs. □

6 Related Work

Higher-order model checking. Model-checking recursion schemes against modal μ-calculus (known as higher-order model checking) has been proved to be decidable by Ong [15], and applied to various verification problems of higher-order programs [7, 11, 12, 17]. The higher-order model-checking problem is n-EXPTIME

complete for order-n recursion schemes [15]. The reachability problem for *call-by-name* programs is an instance of the higher-order model checking, and $(n-1)$-EXPTIME complete for order-n programs [10].

Model-checking call-by-value programs via the CPS translation. The previous approach for model-checking call-by-value programs is based on the CPS translation. Our result implies that the upper bound given by the CPS translation is not tight. However this does not imply that the CPS translation followed by call-by-name model-checking is inefficient. It depends on the model-checking algorithm. For example, the naïve algorithm in [7] following the CPS translation takes more time than our algorithm, but we conjecture that HorSat [2] following the CPS translation meets the tight bound.

Sato et al. [16] employed the selective CPS translation [14] to avoid unnecessary growth of the order, using a type and effect system to capture effect-free fragments and then added continuation parameters to only effectful parts.

Intersection types for call-by-value calculi. Davies and Pfenning [3] studied an intersection type system for a call-by-value effectful calculus and pointed out that the value restriction on the intersection introduction rule is needed. In our type system, the intersection introduction rule is restricted immediately after the abstraction rule, which can be considered as a variant of the value restriction.

Similarly to the previous work on type-based approaches for higher-order model checking [7–9], our intersection type system is a variant of the Essential Type Assignment System in the sense of van Bakel [18], in which the typing rules are syntax directed. Our syntax of intersection types differs from the standard one for call-by-name calculi. Our syntax is inspired by the embedding of the call-by-value calculus into the linear lambda calculus [13], in which the call-by-value function type $A \to B$ is translated into $!(A \multimap B)$ (recall that function types in our intersection type system is $\bigwedge_i(\tau_i \to \sigma_i)$).

Zeilberger [19] proposed a principled design of the intersection type system based on the idea from focusing proofs [1]. Its connection to ours is currently unclear, mainly because of the difference of the target calculi.

Our type system is designed to be complete. This is a characteristic feature that the previous work for call-by-value calculi [3, 19] does not have.

7 Conclusion

We have studied the complexity of the reachability problem for call-by-value programs, and proved the following results. First, the reachability problem for order-3 programs is non-elementary, and thus the order of the program does not serve as a good measure of the complexity, in contrast to the call-by-name case. Second, the reachability problem for depth-n programs is n-EXPTIME complete, which improves the previous upper bound given by the CPS translation.

For future work, we aim to (1) develop an efficient model-checker for call-by-value programs, using the type system proposed in the paper, and (2) study the relationship between intersection types and focused proofs [1, 19].

Acknowledgement This work is partially supported by JSPS KAKENHI Grant Number 23220001.

References

1. J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.
2. C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL 2013*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
3. R. Davies and F. Pfenning. Intersection types and computational effects. In *ICFP 2000*, pages 198–208. ACM, 2000.
4. J. Engelfriet. Iterated stack automata and complexity classes. *Inf. Comput.*, 95(1):21–75, 1991.
5. A. Igarashi and N. Kobayashi. Resource usage analysis. *ACM Trans. Program. Lang. Syst.*, 27(2):264–313, 2005.
6. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.
7. N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL 2009*, pages 416–428. ACM, 2009.
8. N. Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20, 2013.
9. N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS 2009*, pages 179–188. IEEE Computer Society, 2009.
10. N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.
11. N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and CEGAR for higher-order model checking. In *PLDI 2011*, pages 222–233. ACM, 2011.
12. N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *POPL 2010*, pages 495–508. ACM, 2010.
13. J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Electr. Notes Theor. Comput. Sci.*, 1:370–392, 1995.
14. L. R. Nielsen. A selective CPS transformation. *Electr. Notes Theor. Comput. Sci.*, 45:311–331, 2001.
15. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society, 2006.
16. R. Sato, H. Unno, and N. Kobayashi. Towards a scalable software model checker for higher-order programs. In *PEPM 2013*, pages 53–62. ACM, 2013.
17. Y. Tobita, T. Tsukada, and N. Kobayashi. Exact flow analysis by higher-order model checking. In *FLOPS 2012*, volume 7294 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2012.
18. S. van Bakel. Intersection type assignment systems. *Theor. Comput. Sci.*, 151(2):385–435, 1995.
19. N. Zeilberger. Refinement types and computational duality. In *PLPV 2009*, pages 15–26. ACM, 2009.

A Proof of Lemma 1 (Termination)

By the standard technique based on the logical relation. For every sort κ, we define a set of closed terms R_κ by the following rules.

- $t \in R_{\mathbb{B}}$ if and only if
  1. $\emptyset \vdash t :: \mathbb{B}$, and
  2. t has no infinite reduction sequence.
- $t \in R_{\kappa_1 \times \ldots \times \kappa_n \to \iota}$ if and only if
  1. $\emptyset \vdash t :: \kappa_1 \times \ldots \times \kappa_n \to \iota$,
  2. t has no infinite reduction sequence, and
  3. $\forall u_1 \in R_{\kappa_1} \ldots u_n \in R_{\kappa_n}.\ t\,\langle u_1, \ldots, u_n \rangle \in R_\iota$.

Our goal is to prove that $\emptyset \vdash t :: \kappa$ implies $t \in R_\kappa$. For a sequence of terms $\vec{t} = t_1, \ldots, t_n$ and a sequence of sorts $\vec{\kappa} = \kappa_1, \ldots, \kappa_n$, we write $\emptyset \vdash \vec{t} :: \vec{\kappa}$ just if $\emptyset \vdash t_i :: \kappa_i$ for every $i \in \{1, \ldots, n\}$ and $\vec{t} \in R_{\vec{\kappa}}$ just if $u_i \in R_{\kappa_i}$ for every i. We write $[\vec{v}/\vec{x}]t$ for simultaneous substitution of x_i in t to v_i for all i.

Lemma 9. *$\vec{x} :: \vec{\kappa} \vdash t :: \iota$ and $\emptyset \vdash \vec{u} :: \vec{\kappa}$ implies $\emptyset \vdash [\vec{u}/\vec{x}]t :: \iota$.*

Proof. By induction on t. □

Lemma 10. *Suppose that $\emptyset \vdash t :: \kappa$ and $\{t_1, \ldots, t_n\}$ be a set of terms that satisfies the following condition:*

$$t \longrightarrow^* t' \text{ implies } t' \longrightarrow^* t_i \text{ or } t_i \longrightarrow^* t' \text{ for some } i \in \{1, \ldots, n\}.$$

If $t_i \in R_\kappa$ for every i, then $t \in R_\kappa$.

Proof. By induction on the sort κ. □

Lemma 11. *Suppose that $\vec{x} :: \vec{\kappa} \vdash t :: \iota$ and $\vec{v} \in R_{\vec{\kappa}}$, where $\vec{v}$ is a sequence of values. Then $[\vec{v}/\vec{x}]t \in R_\iota$.*

Proof. By induction on the structure of t. We prove the case that $t = \lambda\langle\vec{y}\rangle.u$. Other cases can be proved easily.

Assume that $\vec{v} \in R_{\vec{\kappa}}$ and $\iota = \vec{\kappa}' \to \iota'$. It is necessary to prove that $[\vec{v}/\vec{x}](\lambda\langle\vec{y}\rangle.u) = \lambda\langle\vec{y}\rangle.([\vec{v}/\vec{x}]u) \in R_{\vec{\kappa}' \to \iota'}$. Since $\lambda\langle\vec{y}\rangle.([\vec{v}/\vec{x}]u)$ is a value and cannot be reduced more, it suffices to prove the condition (3). Let $\vec{s} = s_1, \ldots, s_n$ be a sequence of terms such that $\vec{s} \in R_{\vec{\kappa}'}$. Let $t' = [\vec{v}/\vec{x}]t$ and

$$A_1 = \{t'\,\langle\vec{s}'\rangle \mid t'\,\langle\vec{s}\rangle \longrightarrow^* t'\,\langle\vec{s}'\rangle \text{ and } t'\,\langle\vec{s}'\rangle \text{ cannot be reduced more}\}$$
$$A_2 = \{[\vec{w}/\vec{y}, \vec{v}/\vec{x}]u \mid t'\,\langle\vec{s}\rangle \longrightarrow^* t'\,\langle\vec{w}\rangle \text{ where } \vec{w} \text{ is a sequence of values}\}.$$

For every $t'\,\langle\vec{s}'\rangle \in A_1$, we have $t'\,\langle\vec{s}'\rangle \in R_{\iota'}$ since if $t'\,\langle\vec{s}'\rangle$ gets stuck, then $t'\,\langle\vec{s}'\rangle\,\langle\vec{s}''\rangle$ also gets stuck for every $\vec{s}''$. For every $t'' \in A_2$, we have $t'' \in R_{\iota'}$ by the induction hypothesis and $\vec{w} \in R_{\vec{\kappa}'}$. The set of terms $A_1 \cup A_2$ satisfies the condition of Lemma 10, and thus $t'\,\vec{s} \in R_{\iota'}$. □

Lemma 1 is a special case of Lemma 11, in which $\vec{x}$ is the empty sequence.

B Proof of Lemma 2

We first introduce an ordering on terms, defined by induction on terms:

$$
\begin{array}{rcl}
\Omega \preceq t & & \\
x \preceq x & & \\
\lambda\langle\vec{x}\rangle.t \preceq \lambda\langle\vec{x}\rangle.u & \text{iff} & t \preceq u \\
t\,\langle\vec{u}\rangle \preceq t'\,\langle\vec{u}'\rangle & \text{iff} & t \preceq t' \text{ and } u_i \preceq u'_i \text{ for all } i \\
t_1 \oplus t_2 \preceq u_1 \oplus u_2 & \text{iff} & t_1 \preceq u_1 \text{ and } t_2 \preceq u_2 \\
\mathtt{t} \preceq \mathtt{t} & & \\
\mathtt{f} \preceq \mathtt{f} & & \\
\mathbf{if}(t_1,t_2,t_3) \preceq \mathbf{if}(u_1,u_2,u_3) & \text{iff} & t_1 \preceq u_1 \text{ and } t_2 \preceq u_2 \text{ and } t_3 \preceq u_3 \\
\mathfrak{F}_\kappa \preceq \mathfrak{F}_\kappa & &
\end{array}
$$

Lemma 12.

1. *$\preceq$ is a partial order.*
2. *$t \preceq t'$ and $t \longrightarrow^* E[\mathfrak{F}]$ implies $t' \longrightarrow^* E'[\mathfrak{F}]$.*
3. *For every function definition D, $n \leq m$ implies $[t]^n_D \preceq [t]^m_D$.*

Proof. (1): Easy.

(2): First we prove that $E[\mathfrak{F}] \preceq t'$ implies $t' = E'[\mathfrak{F}]$ by induction on E. Second we prove that $t \preceq t'$ and $t \longrightarrow u$ implies $t' \longrightarrow u'$ and $u \preceq u'$ for some u'. The proposition (2) is a consequence of these facts.

(3): By induction on n and t. □

Lemma 13. *Let D be a function definition and t be a term and $n > 0$. If $t \longrightarrow_D u$, then there exists t' such that $[t]^n_D \longrightarrow t'$ or $[t]^n_D = t'$ and $[u]^{n-1}_D \preceq t'$.*

Proof. If $t = E[f]$ for some function symbol f, then $u = E[D(f)]$. For a given evaluation context E and E', the nth approximation $[E]^n_D$ and the ordering $E \preceq E'$ are defined straightforwardly. It is easy to see the following properties:

- $E \preceq E'$ implies $E[t] \preceq E'[t]$ for every t.
- $[E]^n_D \preceq [E]^m_D$ if $n \leq m$.

Then

$$[t]^n_D = [E]^n_D[[f]^n_D]$$

and

$$[u]^{n-1}_D = [E]^{n-1}_D[[D(f)]^{n-1}_D] = [E]^{n-1}_D[[f]^n_D].$$

Set $t' = [t]^n_D$ and we have $[u]^{n-1}_D \preceq t'$ as required. Other cases can be proved easily. □

Let t be a term without any function symbols and u be a term containing function symbols. We write $t \preceq_D u$ just if $t \preceq [u]^n_D$ for some n.

Lemma 14. *If $t \preceq_D u$ and $t \longrightarrow t'$, then $u \longrightarrow^*_D u'$ and $t' \preceq_D u'$ for some u'.*

Proof. By induction on the structure of t. □

Lemma 15. *If $E[\mathfrak{F}] \preceq_D u$, then $u = E'[\mathfrak{F}]$.*

Proof. By induction on E.

Proof (Proof of Lemma 2). Let $P = \texttt{let rec } D \texttt{ in } t$ be a program.

We prove that $t \longrightarrow_D^* E[\mathfrak{F}]$ implies $[t]_D^n \longrightarrow^* E'[\mathfrak{F}]$ for some n, by induction on the length of $t \longrightarrow_D^* E[\mathfrak{F}]$. The base case, where $t = E[\mathfrak{F}]$, can be proved by induction on E. Assume that $t \longrightarrow_D u \longrightarrow_D^* E[\mathfrak{F}]$. Then by the induction hypothesis, we have $[u]_D^n \longrightarrow^* E'[\mathfrak{F}]$ for some n. By Lemma 13, we have t' such that

- $[t]_D^{n+1} \longrightarrow t'$ or $[t]_D^{n+1} = t'$, and
- $[u]_D^n \preceq t'$.

By Lemma 12(2), we have $t' \longrightarrow^* E''[\mathfrak{F}]$. Thus $[t]_D^{n+1} \longrightarrow^* E''[\mathfrak{F}]$.

We prove the converse. Assume that $[t]_D^n \longrightarrow^* E[\mathfrak{F}]$ for some n. More precisely,

$$[t]_D^n \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_n = E[\mathfrak{F}].$$

By definition of $\preceq_D$, we have $[t]_D^n \preceq_D t$. Then by Lemma 14, we have

$$t \longrightarrow_D^* u_1 \longrightarrow_D^* u_2 \longrightarrow_D^* \ldots \longrightarrow_D^* u_n$$

where $t_i \preceq_D u_i$ for every i. In particular, $t \longrightarrow_D^* u_n$ and $E[\mathfrak{F}] \preceq_D u_n$. By Lemma 15, we have $u_n = E'[\mathfrak{F}]$ as desired. □

C Definitions, Proofs of Lemmas in Section 3

C.1 Definition of Replacement

Let $\mathbf{N} = (N, D, \iota, \{V_i\}_{i\in\{0,1,\ldots,N-1\}}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$ be an implementation of natural numbers up to N.

For a sort κ possibly containing the natural number sort, $\kappa^{\mathbf{N}}$ is defined by:

$$\mathbb{B}^{\mathbf{N}} = \mathbb{B}$$
$$\mathbb{N}^{\mathbf{N}} = \iota$$
$$(\kappa_1 \times \ldots \times \kappa_n \to \delta)^{\mathbf{N}} = \kappa_1^{\mathbf{N}} \times \cdots \times \kappa_n^{\mathbf{N}} \to \delta^{\mathbf{N}}.$$

For a term t with natural numbers up to N, $t^{\mathbf{N}}$ is inductively defined by:

$$\begin{aligned}
\underline{0}^{\mathbf{N}} &= \mathbf{z} \\
\mathtt{S}^{\mathbf{N}} &= \mathbf{s} \\
\mathtt{P}^{\mathbf{N}} &= \mathbf{p} \\
\mathtt{EQ}^{\mathbf{N}} &= \mathbf{eq} \\
x^{\mathbf{N}} &= x \\
f^{\mathbf{N}} &= f \\
(\lambda\langle\vec{x}\rangle.t)^{\mathbf{N}} &= \lambda\langle\vec{x}\rangle.(t^{\mathbf{N}}) \\
(t\,\langle u_1, \ldots, u_n\rangle)^{\mathbf{N}} &= (t^{\mathbf{N}})\,\langle u_1^{\mathbf{N}}, \ldots, u_n^{\mathbf{N}}\rangle \\
(t \oplus u)^{\mathbf{N}} &= (t^{\mathbf{N}}) \oplus (u^{\mathbf{N}}) \\
\mathtt{t}^{\mathbf{N}} &= \mathtt{t} \\
\mathtt{f}^{\mathbf{N}} &= \mathtt{f} \\
\mathbf{if}(s,t,u)^{\mathbf{N}} &= \mathbf{if}(s^{\mathbf{N}}, t^{\mathbf{N}}, u^{\mathbf{N}}) \\
\mathfrak{F}^{\mathbf{N}} &= \mathfrak{F} \\
\Omega^{\mathbf{N}} &= \Omega
\end{aligned}$$

Let $D' = \{f_i = \lambda\langle\vec{x}_i\rangle.t_i \mid i \in I\}$ be a function definition. Then $(D')^{\mathbf{N}}$ is defined by:

$$(\{f_i = \lambda\langle\vec{x}_i\rangle.t_i \mid i \in I\})^{\mathbf{N}} = \{f_i = \lambda\langle\vec{x}_i\rangle.(t_i^{\mathbf{N}}) \mid i \in I\} \cup D.$$

For a sort environment Δ, we define

$$\Delta^{\mathbf{N}} = \{x :: \kappa^{\mathbf{N}} \mid x :: \kappa \in \Delta\}.$$

Lemma 16. *If $\Delta \mid \mathcal{K} \vdash t :: \kappa$, then $\Delta^{\mathbf{N}} \mid \mathcal{K}^{\mathbf{N}} \vdash t^{\mathbf{N}} :: \kappa^{\mathbf{N}}$. Therefore if $P =$ $\mathtt{let\ rec}\ D'\ \mathtt{in}\ t$ is a well-sorted program with natural numbers, then $P^{\mathbf{N}} =$ $\mathtt{let\ rec}\ (D')^{\mathbf{N}}\ \mathtt{in}\ t^{\mathbf{N}}$ is a well-sorted program without natural numbers.*

Proof. By induction on t. □

C.2 Proof of Lemma 4

Let $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ be a program with natural numbers up to N, and $\mathbf{N} = (N, D_{\mathbf{N}}, \kappa, \{V_i\}_{i\in\{0,1,\ldots,N-1\}}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z})$ be an implementation of natural numbers up to N.

First we prove the left-to-right direction. Assume $\mathtt{let\ rec}\ D\ \mathtt{in}\ t$ fails. We prove that $\mathtt{let\ rec}\ D^{\mathbf{N}}\ \mathtt{in}\ t^{\mathbf{N}}$ fails.

A simulation $\sim$ between terms with natural numbers and terms without natural numbers is defined by:

$$\begin{array}{rll}
\underline{n} \sim v_n & \text{if} & v_n \in V_n \\
\mathtt{S} \sim \mathbf{s} & & \\
\mathtt{P} \sim \mathbf{p} & & \\
\mathtt{EQ} \sim \mathbf{eq} & & \\
x \sim x & & \\
t\,\langle u_1, \ldots, u_n \rangle \sim t'\,\langle u'_1, \ldots, u'_n \rangle & \text{if} & t \sim t' \text{ and } u_i \sim u'_i \text{ for every } i \in \{1, \ldots, n\} \\
\lambda\langle \vec{x} \rangle.t \sim \lambda\langle \vec{x} \rangle.u & \text{if} & t \sim u \\
t_1 \oplus t_2 \sim u_1 \oplus u_2 & \text{if} & t_1 \sim u_1 \text{ and } t_2 \sim u_2 \\
\mathtt{t} \sim \mathtt{t} & & \\
\mathtt{f} \sim \mathtt{f} & & \\
\mathbf{if}(t_0, t_1, t_2) \sim \mathbf{if}(u_0, u_1, u_2) & \text{if} & t_i \sim u_i \text{ for } i = 0, 1, 2 \\
\mathfrak{F} \sim \mathfrak{F} & & \\
\Omega \sim \Omega & &
\end{array}$$

We write $D \sim D'$ just if

1. $\mathrm{dom}(D') = \mathrm{dom}(D) \uplus \mathrm{dom}(D_{\mathbf{N}})$,
2. $D(f) \sim D'(f)$ for every $f \in \mathrm{dom}(D)$, and
3. $D'(f) = D_{\mathbf{N}}(f)$ for every $f \in \mathrm{dom}(D_{\mathbf{N}})$.

The simulation relation on evaluation contexts is defined by the rules above and the following rule:

$$\Box \sim \Box.$$

Lemma 17. *For every term t with natural numbers, we have $t \sim t^{\mathbf{N}}$. Similarly, for every function definition D with natural numbers, we have $D \sim D^{\mathbf{N}}$.*

Proof. The first part can be proved by induction on the structure of t. The second part can be proved by using the first part. □

Lemma 18.

1. *If $v \sim u$ for some value v, then u is a value.*
2. *If $E[t'] \sim u$, there exist unique E' and u' such that $u = E'[u']$ and $E \sim E'$ and $t' \sim u'$.*
3. *If $E[\mathfrak{F}] \sim u$, then $u = E'[\mathfrak{F}]$ for some evaluation context E'.*
4. *If $t \sim v$ for some value v, then t is a value.*
5. *If $t \sim E'[u']$, there exist unique E and t' such that $t = E[t']$ and $E \sim E'$ and $t' \sim u'$.*
6. *If $t \sim E'[\mathfrak{F}]$, then $t = E[\mathfrak{F}]$ for some evaluation context E.*

Proof.

(1): Since v is a value, we know that $v = \lambda\langle \vec{x} \rangle.t$, $\mathtt{t}$, $\mathtt{f}$, $\mathtt{S}$, $\mathtt{P}$, $\mathtt{EQ}$ or $\underline{n}$. It is easy to check that u is a value in every case.

(2): By induction on the structure of E.

(3): By induction on the structure of E, using (1).

(4): Since v is a value, we know that $v = \lambda\langle\vec{x}\rangle.t$, $\mathtt{t}$ or $\mathtt{f}$. If $v = \lambda\langle\vec{x}\rangle.u$, then $t = \lambda\langle\vec{x}\rangle.t'$, $\mathtt{S}$, $\mathtt{P}$, $\mathtt{EQ}$ or $\underline{n}$, and thus t is a value. If $v = \mathtt{t}$ (resp. $\mathtt{f}$), then $t = \underline{n}$ or $\mathtt{t}$ (resp. $\mathtt{f}$), and thus t is a value.

(5): By induction on the structure of E'.

(6): By induction on the structure of E', using (4). □

Lemma 19. *If $t \sim t'$ and $u_i \sim u'_i$ for every $i \in \{1, \ldots, n\}$, then $[\vec{u}/\vec{x}]t \sim [\vec{u'}/\vec{x}]t'$. If $E \sim E'$ and $t \sim t'$, then $E[t] \sim E'[t']$.*

Proof. The first proposition is proved by induction on t. The second proposition is proved by induction on E. □

Lemma 20. *Assume that $t \sim t'$ and $D \sim D'$. If $t \longrightarrow_D^* E[\mathfrak{F}]$, then $t' \longrightarrow_{D'}^* E'[\mathfrak{F}]$.*

Proof. By induction on the length of $t \longrightarrow_D^* E[\mathfrak{F}]$. If $t = E[\mathfrak{F}]$, then by Lemma 18(3), we have $t' = E'[\mathfrak{F}]$. Assume that $t \longrightarrow_D u \longrightarrow_D^* E[\mathfrak{F}]$. It suffices to prove that $t' \longrightarrow_{D'}^* u'$ and $u \sim u'$ for some u'.

Case $t = E_0[\mathtt{S}\,\underline{n}] \longrightarrow_D E_0[\underline{n+1}] = u$: Then by Lemma 18(2), there exist E'_0 and t'' such that $t' = E'_0[t'']$ and $E_0 \sim E'_0$ and $\mathtt{S}\,\underline{n} \sim t''$. Hence $t'' = \mathbf{s}\,v_n$ for some $v_n \in V_n$. Then $t'' = \mathbf{s}\,v_n \longrightarrow_{D_\mathbf{N}}^* v_{n+1}$ for some $v_{n+1} \in V_{n+1}$, which implies $t' = E'_0[t''] \longrightarrow_{D'}^* E'_0[v_{n+1}]$. Since $\underline{n+1} \sim v_{n+1}$, from Lemma 19, we have $u = E_0[\underline{n+1}] \sim E'_0[v_{n+1}]$.

Case $t = E_0[(\lambda\langle\vec{x}\rangle.t_0)\,\langle\vec{v}\rangle] \longrightarrow_D E_0[\,[\vec{v}/\vec{x}]t_0\,] = u$: Then by Lemma 18(2), there exist E'_0 and t'' such that $t' = E'_0[t'']$ and $E_0 \sim E'_0$ and $(\lambda\langle\vec{x}\rangle.t_0)\,\langle\vec{v}\rangle \sim t''$. Then $t'' = (\lambda\langle\vec{x}\rangle.t'_0)\,\langle\vec{v'}\rangle$ with $t_0 \sim t'_0$ and $v_i \sim v'_i$ for every i. We have

$$E'_0[(\lambda\langle\vec{x}\rangle.t'_0)\,\langle\vec{v'}\rangle] \longrightarrow_{D'} E'_0[\,[\vec{v'}/\vec{x}]t'_0\,].$$

By Lemma 19, we have $[\vec{v}/\vec{x}]t_0 \sim [\vec{v'}/\vec{x}]t'_0$. So by Lemma 19, we have $u = E_0[\,[\vec{v}/\vec{x}]t_0\,] \sim E'_0[\,[\vec{v'}/\vec{x}]t'_0\,]$.

Case $t = E_0[f] \longrightarrow_D E_0[D(f)] = u$: By Lemma 18(2), there exist E'_0 and t'' such that $t' = E'_0[t'']$ and $E_0 \sim E'_0$ and $f \sim t''$. Hence $t'' = f$. Now we have $t' = E'_0[f] \longrightarrow_{D'} E'_0[D'(f)]$. From the assumption, we know that $D(f) \sim D'(f)$. So by Lemma 19, we have $u = E_0[D(f)] \sim E'_0[D'(f)]$ as required.

Other cases can be proved similarly. □

Lemma 21. *Assume that $t \sim t'$ and $D \sim D'$. If $t' \longrightarrow_{D'}^* E'[\mathfrak{F}]$, then $t \longrightarrow_D^* E[\mathfrak{F}]$.*

Proof. By induction on the length of $t' \longrightarrow_{D'}^* E'[\mathfrak{F}]$. If $t' = E'[\mathfrak{F}]$, then by Lemma 18(6), we have $t = E[\mathfrak{F}]$. Assume that $t' = E'_0[t'_0] \longrightarrow_{D'} u' \longrightarrow_{D'}^* E'[\mathfrak{F}]$, where t'_0 is the redex. By Lemma 18(5), there exist E_0 and t_0 such that $E_0 \sim E'_0$ and $t_0 \sim t'_0$ and $t = E_0[t_0]$. We prove that there exists u'' and u such that

- $t' \longrightarrow_{D'} u' \longrightarrow_{D'}^* u'' \longrightarrow_{D'} E'[\mathfrak{F}]$,
- $t \longrightarrow_D u$, and
- $u \sim u''$.

If so, by the induction hypothesis, we have $t \longrightarrow_D u \longrightarrow_D^* E[\mathfrak{F}]$ as required. We prove the above claim by case analysis of t'_0.

Consider the case that $t'_0 = (\lambda\langle\vec{x}\rangle.t'_{00})\,\langle\vec{v}'\rangle$. We do case analysis of t_0. If $t_0 = (\lambda\langle\vec{x}\rangle.t_{00})\,\langle\vec{v}\rangle$, it is easy to prove the claim. Assume that $t_0 = \mathtt{S}\,\underline{n}$. Since $t_0 = \mathtt{S}\,\underline{n} \sim (\lambda\langle\vec{x}\rangle.t'_{00})\,\langle\vec{v}'\rangle$, we know that $\lambda\langle\vec{x}\rangle.t'_{00} = \mathbf{s}$ and $\vec{v}' = v_n$ for some $v_n \in V_n$ (i.e. the length of the sequence $\vec{v}'$ is 1). Since evaluation of $\mathbf{s}\,v_n$ does not fail, the reduction sequence $t' \longrightarrow_{D'}^* E'[\mathfrak{F}]$ can be decomposed as follows:

$$t' = E'_0[\mathbf{s}\,v_n] \longrightarrow_{D'}^* E'_0[v_{n+1}] \longrightarrow_{D'}^* E'[\mathfrak{F}],$$

where $v_{n+1} \in V_{n+1}$ (so $n+1 < N$). Then we have $E_0[\mathtt{S}\,\underline{n}] \longrightarrow_D E_0[\underline{n+1}]$ and $E_0[\underline{n+1}] \sim E'_0[v_{n+1}]$ as desired.

Other cases can be proved similarly. □

Proof (Proof of Lemma 4). Lemma 4 is a direct consequence of Lemma 17, Lemma 20 and Lemma 21. □

C.3 Definition of ${}^{\mathbb{B}}\mathbf{N}$ and Proof of Lemma 5

Let $\mathbf{N} = (N, D, \kappa, \{V_i\}_{i\in\{0,1,\ldots,N-1\}}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$ be an implementation of natural numbers up to N. First we define the implementation ${}^{\mathbb{B}}\mathbf{N} = (2^N, D \cup D', \mathbb{B} \to \kappa, \{V'_i\}_{i\in\{0,1,\ldots,2^N-1\}}, \mathbf{eq}', \mathbf{s}', \mathbf{p}', \mathbf{z}', \mathbf{max}')$ of natural numbers up to 2^N.

- The additional function definition D' consists of:

$$\begin{aligned}
\mathtt{up} &= \lambda i^{\kappa}.(i \oplus (\mathtt{up}(\mathbf{s}\,i)))\\
\mathtt{get} &= \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}\rangle.\quad \mathbf{if}(\mathbf{eq}\,\langle x\,\mathtt{t}, i\rangle,\ \mathtt{t},\ \Omega)\ \oplus\ \mathbf{if}(\mathbf{eq}\,\langle x\,\mathtt{f}, i\rangle,\ \mathtt{f},\ \Omega)\\
\mathtt{put} &= \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}, b^{\mathbb{B}}\rangle.\lambda c^{\mathbb{B}}.\ \big(\mathbf{if}(b = c, i, \Omega)\ \oplus\ ((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle i, j\rangle, \Omega, j))\,(x\,c))\big)\\
\mathtt{ss} &= \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}\rangle.\quad \mathbf{if}(\mathtt{get}\,\langle x, i\rangle,\quad \mathtt{ss}\,\langle \mathtt{put}\,\langle x, i, \mathtt{f}\rangle, \mathbf{s}\,i\rangle,\qquad \mathtt{put}\,\langle x, i, \mathtt{t}\rangle)\\
\mathtt{pp} &= \lambda\langle x^{\mathbb{B}\to\kappa}, i^{\kappa}\rangle.\quad \mathbf{if}(\mathtt{get}\,\langle x, i\rangle,\quad \mathtt{put}\,\langle x, i, \mathtt{f}\rangle,\qquad \mathtt{pp}\,\langle \mathtt{put}\,\langle x, i, \mathtt{t}\rangle, \mathbf{s}\,i\rangle)\\
e &= \lambda\langle x^{\mathbb{B}\to\kappa}, y^{\mathbb{B}\to\kappa}, i^{\kappa}\rangle.\\
&\qquad \mathbf{if}(\mathtt{get}\,\langle x, i\rangle = \mathtt{get}\,\langle y, i\rangle,\quad \mathbf{if}(\mathbf{eq}\,\langle i, \mathbf{z}\rangle, \mathtt{t}, e\,\langle x, y, \mathbf{p}\,i\rangle),\quad \mathtt{f})
\end{aligned}$$

- Let $m < 2^N$ and $b_{N-1}\ldots b_0$ be its binary representation. Then V'_m is the set of values v of sort $\mathbb{B} \to \kappa$ such that
  1. $b_i = 1$ iff $v\,\mathtt{t} \longrightarrow^* v'$ for some $v' \in V_i$,
  2. $b_i = 0$ iff $v\,\mathtt{f} \longrightarrow^* v'$ for some $v' \in V_i$, and
  3. $v\,\mathtt{t} \longrightarrow^* v'$ or $v\,\mathtt{f} \longrightarrow^* v'$ implies $v' \in \bigcup_{i\in\{0,\ldots,N-1\}} V_i$.
- $\mathbf{eq}' = \lambda x^{\mathbb{B}\to\kappa}.\lambda y^{\mathbb{B}\to\kappa}.e\langle x, y, \mathbf{max}\rangle$.
- $\mathbf{s}' = \lambda x^{\mathbb{B}\to\kappa}.\mathtt{ss}\langle x, \mathbf{z}\rangle$ and $\mathbf{p}' = \lambda x^{\mathbb{B}\to\kappa}.\mathtt{pp}\langle x, \mathbf{z}\rangle$.
- $\mathbf{z}' = \lambda x^{\mathbb{B}}.\mathbf{if}(x, \Omega, \mathtt{up}\,\mathbf{z})$.
- $\mathbf{max}' = \lambda x^{\mathbb{B}}.\mathbf{if}(x, \mathtt{up}\,\mathbf{z}, \Omega)$.

We prove that ${}^{\mathbb{B}}\mathbf{N}$ is an implementation of natural numbers up to 2^N. Set $V = \bigcup_i V_i$ and $V' = \bigcup_i V'_i$. In this subsection, we simply write $\longrightarrow$ for $\longrightarrow_{D \cup D'}$.

For a natural number $k < 2^N$, the *binary encoding of* k is a sequence $b_{N-1}b_{N-2}\dots b_1b_0$ of Boolean values of length N such that

$$k = \sum_{\substack{l \in [0, N-1] \\ b_l = \mathtt{t}}} 2^l$$

We write $k = [b_{N-1}b_{N-2}\dots b_1b_0]$ if $b_{N-1}b_{N-2}\dots b_1b_0$ is the binary encoding of k.

Lemma 22. *Let $v \in V_k$.*

1. *For every k' such that $k \leq k' < N$, $\mathtt{up}\, v \longrightarrow^* w \in V_{k'}$ for some w.*
2. *If $\mathtt{up}\, v \longrightarrow^* w$, then $w \in V$.*

Proof. By induction on $N - k$.

(Case $k = N - 1$) To prove (1), consider the following reduction sequence:

$$\mathtt{up}\, v \longrightarrow v \oplus \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow v \in V_{N-1}.$$

This gives a witness of (1) since k' must be $N - 1$. We prove (2). Assume that $\mathtt{up}\, v \longrightarrow^* w$ and $w \notin V$. Since $t = v \oplus \mathtt{up}\, (\mathbf{s}\, v)$ is the only term such that $\mathtt{up}\, v \longrightarrow t$, we have $t \longrightarrow^* w$. Since $w \neq v \in V_{N-1}$, we have $t \longrightarrow \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow^* w$. However $\mathbf{s}\, v$ does not converge to any value, which contradicts the assumption.

(Case $k < N - 1$) We prove (1). We know that $k' = k$ or $k' > k$. For the former case, we have

$$\mathtt{up}\, v \longrightarrow v \oplus \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow v \in V_k.$$

For the latter case, we have $\mathbf{s}\, v \longrightarrow^* v' \in V_{k+1}$ for some v' and

$$\mathtt{up}\, v \longrightarrow v \oplus \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow^* \mathtt{up}\, v' \longrightarrow^* w \in V_{k'},$$

where we use the induction hypothesis for the last reduction step. We prove (2). Assume that $\mathtt{up}\, v \longrightarrow^* w$. Then we have

$$\mathtt{up}\, v \longrightarrow v \oplus \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow v = w$$

or

$$\mathtt{up}\, v \longrightarrow v \oplus \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow \mathtt{up}\, (\mathbf{s}\, v) \longrightarrow^* \mathtt{up}\, v' \longrightarrow^* w.$$

For the former case, we have $w \in V_k \subseteq V$ as desired. For the latter case, we have $w \in V$ by the induction hypothesis. □

Lemma 23. *Let $v \in V'_k$, $w_i \in V_i$ and $b \in \{\mathtt{t}, \mathtt{f}\}$. Assume that*

$$k = [b_{N-1}b_{N-2}\dots b_1b_0].$$

Then $\mathtt{get}\, \langle v, w_i \rangle \longrightarrow^* b$ if and only if $b_i = b$.

Proof. First we prove the left-to-right direction. Assume that $\mathtt{get}\,\langle v, w_i\rangle \longrightarrow^* b$. Then we have

$$\begin{aligned}\mathtt{get}\,\langle v, w_i\rangle &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{t}, w_i\rangle, \mathtt{t}, \Omega) \;\oplus\; \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{t}, w_i\rangle, \mathtt{t}, \Omega)\\ &\longrightarrow^* b\end{aligned}$$

or

$$\begin{aligned}\mathtt{get}\,\langle v, w_i\rangle &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{t}, w_i\rangle, \mathtt{t}, \Omega) \;\oplus\; \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow^* b.\end{aligned}$$

Both cases can be proved similarly. Assume the latter case. Then we have $b = \mathtt{f}$. By inspecting the reduction sequence, we have

$$\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle \longrightarrow^* \mathbf{eq}\langle v', w_i\rangle \longrightarrow^* \mathtt{t}$$

with $v\,\mathtt{f} \longrightarrow^* v'$. By definition of V'_k, we have $v' \in V_{i'}$ for some $i' \in \{0, \ldots, N-1\}$. By the condition on $\mathbf{eq}$, we have $i = i'$. Thus we have $v\,\mathtt{f} \longrightarrow^* v' \in V_i$. So by the definition of V'_k, we have $b_i = \mathtt{f}$ as required.

Second we prove the right-to-left direction. Assume that $b_i = b = \mathtt{f}$. Then by definition of V'_k, we have $v\,\mathtt{f} \longrightarrow^* w'_i$ for some $w'_i \in V_i$. Thus

$$\begin{aligned}\mathtt{get}\,\langle v, w_i\rangle &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{t}, w_i\rangle, \mathtt{t}, \Omega) \;\oplus\; \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow \mathbf{if}(\mathbf{eq}\,\langle v\,\mathtt{f}, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow^* \mathbf{if}(\mathbf{eq}\,\langle w'_i, w_i\rangle, \mathtt{f}, \Omega)\\ &\longrightarrow^* \mathbf{if}(\mathtt{t}, \mathtt{f}, \Omega)\\ &\longrightarrow \mathtt{f},\end{aligned}$$

where we use the condition on $\mathbf{eq}$ in the fourth step. The case that $b_i = b = \mathtt{t}$ can be proved similarly. □

Lemma 24. *Let $v \in V'_k$, $w_i \in V_i$ and $b' \in \{\mathtt{t}, \mathtt{f}\}$. Assume that*

$$\begin{aligned}k &= [b_{N-1} b_{N-2} \ldots b_1 b_0]\\ k' &= [b_{N-1} \ldots b_{i+1} b' b_{i-1} \ldots b_0].\end{aligned}$$

Then

- *If* $\mathtt{put}\,\langle v, w_i, b'\rangle \longrightarrow^* v'$, *then* $v' \in V_{k'}$.
- $\mathtt{put}\,\langle v, w_i, b'\rangle \longrightarrow^* v'$ *for some* $v' \in V_{k'}$.

Proof. By definition of $\mathtt{put}$, the only reduction sequence starting from $\mathtt{put}\,\langle v, w_i, b'\rangle$ is

$$\mathtt{put}\,\langle v, w_i, b'\rangle \longrightarrow \lambda c^{\mathbb{B}}.\Big(\mathbf{if}(b' = c, w_i, \Omega) \;\oplus\; \big((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,c)\big)\Big).$$

We write v' for the right-hand-side of the above reduction. In order to prove both items, it suffices to prove that $v' \in V_{k'}$, i.e.,

1. $b_j = \mathtt{t}$ $(j \neq i)$ if and only if $v'\,\mathtt{t} \longrightarrow^* v''$ for some $v'' \in V_j$,
2. $b_j = \mathtt{f}$ $(j \neq i)$ if and only if $v'\,\mathtt{f} \longrightarrow^* v''$ for some $v'' \in V_j$,
3. $b = b'$ if and only if $v'\,b \longrightarrow^* v''$ for some $v'' \in V_i$, and
4. $v'\,\mathtt{t} \longrightarrow^* v''$ or $v'\,\mathtt{f} \longrightarrow^* v''$ implies $v'' \in V$.

(1): The following reduction sequence proves the left-to-right direction:

$$\begin{array}{ll} & v'\,\mathtt{t} \\ \longrightarrow & \mathbf{if}(b' = \mathtt{t}, w_i, \Omega) \oplus \big((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,\mathtt{t})\big) \\ \longrightarrow & (\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,\mathtt{t}) \\ \longrightarrow^* & (\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,w_j \\ \longrightarrow^* & \mathbf{if}(\mathbf{eq}\,\langle w_i, w_j\rangle, \Omega, w_j) \\ \longrightarrow^* & \mathbf{if}(\mathtt{f}, \Omega, w_j) \\ \longrightarrow & v_j, \end{array}$$

where $w_j \in V_j$. We prove the converse. Assume that $v'\,\mathtt{t} \longrightarrow^* w_j$ for some $w_j \in V_j$. The key observation is that the right branch of the non-deterministic branch must be chosen in the reduction sequence, since in the left branch only w_i is returned. Thus the reduction sequence must be of the form:

$$\begin{array}{ll} & v'\,\mathtt{t} \\ \longrightarrow & \mathbf{if}(b' = \mathtt{t}, w_i, \Omega) \oplus \big((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,\mathtt{t})\big) \\ \longrightarrow & (\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,\mathtt{t}) \\ \longrightarrow^* & (\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,w_j \\ \longrightarrow^* & \mathbf{if}(\mathbf{eq}\,\langle w_i, w_j\rangle, \Omega, w_j) \\ \longrightarrow^* & \mathbf{if}(\mathtt{f}, \Omega, w_j) \\ \longrightarrow & w_j. \end{array}$$

So $v\,\mathtt{t} \longrightarrow^* w_j$. Hence $b_j = \mathtt{t}$ by the definition of V_k.

(2): Similar to (1).

(3): The following reduction sequence proves the left-to-right direction:

$$\begin{array}{ll} & v'\,b \\ \longrightarrow & \mathbf{if}(b' = b, w_i, \Omega) \oplus \big((\lambda j.\mathbf{if}(\mathbf{eq}\,\langle w_i, j\rangle, \Omega, j))\,(v\,b)\big) \\ \longrightarrow & \mathbf{if}(b' = b, w_i, \Omega) \\ \longrightarrow^* & \mathbf{if}(\mathtt{t}, w_i, \Omega) \\ \longrightarrow & v_i. \end{array}$$

The converse can also be proved easily.

(4): A consequence of the fact that $v'\,b \longrightarrow^* v''$ implies $v'' = v_i$ or $v\,b \longrightarrow^* v''$. □

Lemma 25. *Assume $v \in V'_k$ and $w_i \in V_i$. Let $k' = k + 2^i$.*

- *If $k' < N$, then $\mathtt{ss}\,\langle v, w_i\rangle \longrightarrow^* v' \in V'_{k'}$ for some v'.*
- *$\mathtt{ss}\,\langle v, w_i\rangle \longrightarrow^* v'$ implies $k' < 2^N$ and $v' \in V'_{k'}$.*

Proof. By induction on $N - i$. □

Lemma 26. *Assume $v \in V'_k$ and $w_i \in V_i$. Let $k' = k - 2^i$.*

- *If $k' \geq 0$, then $\mathtt{pp}\,\langle v, w_i\rangle \longrightarrow^* v' \in V'_{k'}$ for some v'.*
- *$\mathtt{pp}\,\langle v, w_i\rangle \longrightarrow^* v'$ implies $k' \geq 0$ and $v' \in V'_{k'}$.*

Proof. By induction on $N - i$. □

Lemma 27. *Assume $v \in V'_k$, $v' \in V'_{k'}$ and $w_i \in V_i$. Let $k = [b_{N-1}b_{N-2}\ldots b_1 b_0]$ and $k' = [b'_{N-1}b'_{N-2}\ldots b'_1 b'_0]$.*

- *$b_j = b'_j$ for every $j \leq i$ if and only if $e\,\langle v, v', w_i\rangle \longrightarrow^* \mathtt{t}$.*
- *$b_j \neq b'_j$ for some $j \leq i$ if and only if $e\,\langle v, v', w_i\rangle \longrightarrow^* \mathtt{f}$.*

Proof. By induction on $N - i$. □

Lemma 28. *If $\mathbf{N}$ is an implementation of natural numbers up to N, then ${}^{\mathbb{B}}\mathbf{N}$ is an implementation of natural numbers up to 2^N.*

Proof. An easy consequence of Lemma 22, Lemma 23, Lemma 24, Lemma 25, Lemma 26 and Lemma 27. □

Definition of N(0) The implementation $\mathbf{N}(0) = (2, D, \mathbb{B}, \{V_0, V_1\}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$ of natural numbers up to 2 is defined by:

$$D = \{\} \qquad V_0 = \{\mathtt{f}\} \qquad V_1 = \{\mathtt{t}\} \qquad \mathbf{z} = \mathtt{f} \qquad \mathbf{max} = \mathtt{t}$$
$$\mathbf{eq} = \lambda\langle x, y\rangle.\mathbf{if}(x, \mathbf{if}(y, \mathtt{t}, \mathtt{f}), \mathbf{if}(y, \mathtt{f}, \mathtt{t}))$$
$$\mathbf{s} = \lambda x.\mathbf{if}(x, \Omega, \mathtt{t}) \qquad \mathbf{p} = \lambda x.\mathbf{if}(x, \mathtt{f}, \Omega).$$

Lemma 29. $\mathbf{N}(0)$ *is an implementation of natural numbers up to $2 = \mathbf{exp}_0(2)$.*

Proof. Easy. □

Proof (Lemma 5). A consequence of Lemma 28 and Lemma 29.

D Proof of n-EXPTIME Hardness of Depth-n Reachability

D.1 Higher-Order Pushdown Systems

First of all, we define higher-order alternating pushdown systems and related notions. Then we show that the existence of a finite run tree for a given order-n alternating pushdown system is n-EXPTIME complete. We shall reduce this problem to the depth-n reachability problem in the next subsection.

Definition 3 (Higher-Order Stacks). Let Π be a finite set of *stack symbols*. Then the set $\mathcal{S}_n$ of *order-n stacks* are defined by induction on n as follows:

1. $\mathcal{S}_0 = \Pi$, and
2. $\mathcal{S}_{(n+1)} = \mathcal{S}_n^+$.

Thus an order-$(n+1)$ stack is a stack $\alpha_0\alpha_1 \dots \alpha_k$ of order-n stacks, where α_0 is the stack top.

It is convenient to represent an order-$(n+1)$ stack α_{n+1} by a pair of the stack top and the *remainder*, which is a possibly empty sequence of order-n stacks, as $\alpha_{n+1} = \alpha_n : \xi_{n+1}$. By iteratively applying this notation, an order-n stack can be represented as $(\dots((a : \xi_1) : \xi_2) : \dots) : \xi_n$, where a is a stack symbol and ξ_k is an order-k remainder (i.e. a possibly empty sequence of order-$(k-1)$ stacks). We assume : is left-associative, and simply write as $a : \xi_1 : \cdots : \xi_n$.

The set Op_n of operations on order-n stacks is $\{pop_k \mid 1 \leq k \leq n\} \cup \{push_k \mid 1 < k \leq n\} \cup \{push_1^a \mid a \in \Pi\}$.

$$\begin{aligned} pop_k(a_0 : \xi_1 : \dots : \xi_n) &= \xi_k : \dots : \xi_n \\ push_1^{a'}(a : \xi_1 : \dots : \xi_n) &= a' : (a : \xi_1) : \xi_2 : \dots : \xi_n \\ push_k(a_0 : \xi_1 : \dots : \xi_n) &= a_0 : \dots : \xi_{k-1} : (a_0 : \dots : \xi_{k-1} : \xi_k) : \xi_{k+1} : \dots : \xi_n \end{aligned}$$

Note that (i) order-k remainder ξ_k is an order-k stack just if ξ_k is not empty, and (ii) for every order-k stack α_k and order-$(k+1)$ remainder ξ_{k+1}, $\alpha_k : \xi_{k+1}$ is an order-$(k+1)$ remainder (in particular, an order-$(k+1)$ stack). So push operations are always defined but the order-k pop operation is defined just if the order-k remainder ξ_k is not empty.

Given a higher-order stack $\alpha \in \mathcal{S}_n$, its *top symbol* $top(\alpha)$ is defined by:

$$top(a_0 : \xi_1 : \dots : \xi_n) = a_0.$$

Definition 4 (Alternating Higher-Order Pushdown Automaton). An *order-n alternating pushdown automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \Pi, \delta)$, where

- Σ is a finite alphabet,
- Q is a finite set of *states*,
- Π is a finite set of stack symbols, and
- $\delta \subseteq (\Sigma \uplus \{\varepsilon\}) \times Q \times \Pi \times \mathcal{P}(Q \times Op_n)$ is a transition relation, where $\mathcal{P}(A)$ means the powerset of A.

An *order-n alternating pushdown system* (*order-n PDS* for short) is an order-n alternating pushdown automaton over the empty alphabet (i.e. $\Sigma = \emptyset$). For an order-n alternating pushdown system, the transition relation is often considered as a subset of $Q \times \Pi \times \mathcal{P}(Q \times Op_n)$.

We use $\mathcal{O}$ as a metavariable ranging over $\mathcal{P}(Q \times Op_n)$.

Let $\mathcal{A}$ be a pushdown automaton. A *configuration* for $\mathcal{A}$ is a pair $(q, \alpha) \in Q \times \mathcal{S}_n$. The transition relations are naturally extended to configurations. We define the relation $\vDash^{\mathtt{a}}_{\mathcal{A}}$ (here $\mathtt{a} \in \Sigma \uplus \{\varepsilon\}$) on configurations and (finite) sets of configurations by the following rule.

If $(\mathtt{a}, q, a_0, \mathcal{O}) \in \delta$ and $a_0 = top(\alpha)$, then

$$(q, \alpha) \vDash^{\mathtt{a}}_{\mathcal{A}} \{(p, op(\alpha)) \mid (p, op) \in \mathcal{O})\},$$

provided that $op(\alpha)$ is defined for all $(p, op) \in \mathcal{O}$.

Let $w \in \Sigma^*$ be a word and $c = (q, \alpha)$ be a configuration. The relation $w \in \mathcal{L}(q, \alpha)$, read "$w$ is accepted from (q, α)", is inductively defined by the following rules.

1. If $(q, \alpha) \vDash^{\mathtt{a}}_{\mathcal{A}} \emptyset$, then $\mathtt{a} \in \mathcal{L}(q, \alpha)$.
2. If $(q, \alpha) \vDash^{\mathtt{a}}_{\mathcal{A}} \mathcal{C}$ and $w \in \mathcal{L}(p, \beta)$ for every $(p, \beta) \in \mathcal{C}$, then $\mathtt{a}w \in \mathcal{L}(q, \alpha)$.

Note that $\mathtt{a}$ can be the empty word in the above rules. We often write $w \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$ to make the pushdown automaton $\mathcal{A}$ explicit.

Lemma 30 (Engelfriet [4]). *Consider the problem that decides whether $w \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$ for a given order-n alternating higher-order pushdown automaton $\mathcal{A}$, a given word w and a given configuration (q, α). This problem is n-EXPTIME complete.*

If $\mathcal{A}$ is a pushdown system (i.e. $\Sigma = \emptyset$), the above problem becomes simpler since $\mathcal{L}_{\mathcal{A}}(q, \alpha) = \{\ \}$ or $\{\varepsilon\}$. We write $(q, \alpha)\downarrow_{\mathcal{A}}$ just if $\varepsilon \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$. We simply write $(q, \alpha)\downarrow$ if $\mathcal{A}$ is clear from the context. The problem to decide whether $(q, \alpha)\downarrow_{\mathcal{A}}$ is also n-EXPTIME complete for order-n pushdown systems.

Lemma 31. *Consider the problem that decides whether $(q, \alpha)\downarrow_{\mathcal{A}}$ for a given order-n pushdown system $\mathcal{A}$ and a given configuration (q, α). This problem is n-EXPTIME complete.*

Proof. Given an order-n pushdown automaton $\mathcal{A}$, a word w and a configuration (q, α), we construct an order-n pushdown system $\mathcal{A}'$ and a configuration (q', α') in polynomial time such that $w \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$ if and only if $(q', \alpha')\downarrow_{\mathcal{A}'}$. The pushdown system $\mathcal{A}'$ is constructed by embedding words into its states.

Let $\mathcal{A} = (\Sigma, Q, \Pi, \delta)$. For a given word w, we write $Post(w)$ be the set of all postfixes of w, i.e. $Post(w) = \{w' \mid w = w''w' \text{ for some } w''\}$. Formally $\mathcal{A}' = (Q', \Pi', \delta')$ is defined as follows.

- $Q' = Q \times Post(w)$.
- $\Pi' = \Pi$.
- $\delta' \subseteq Q' \times \Pi' \times \mathcal{P}(Q' \times Op_n) = Q \times Post(w) \times \Pi \times \mathcal{P}(Q \times Post(w) \times Op_n)$ is given by:

  If $(\mathtt{a}, q, a_0, \{(p_1, op_1), \ldots, (p_k, op_k)\}) \in \delta$ and $\mathtt{a}w' \in Post(w)$, then one has $(q, \mathtt{a}w', a_0, \{(p_1, w', op_1), \ldots, (p_k, w', op_k)\}) \in \delta'$.

  Note that $\mathtt{a}$ can be the empty word. If so, the word in the states is not changed.

Let $w' \in Post(w)$. Then $w' \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$ if and only if $(q, w', \alpha)\downarrow_{\mathcal{A}'}$. This proposition can be proved by induction on the derivation of $w' \in \mathcal{L}_{\mathcal{A}}(q, \alpha)$ for the left-to-right direction and by induction on the derivation of $(q, w', \alpha)\downarrow_{\mathcal{A}'}$ for the right-to-left direction. □

D.2 Depth-n Program Simulating Order-n PDS

Let $\mathcal{A} = (Q, \Pi, \delta)$ be an order-n PDS (here we omit the alphabet Σ that must be empty by definition). Following [6], we shall prove that every stack operation can be implemented by terms of depth at most n. We assume that $Q = \{1, 2, \ldots, |Q|\}$.

For terms s and t of sort $\mathbb{B}$, we define $s\&t = \mathbf{if}(s, \mathbf{if}(t, \mathtt{t}, \Omega), \Omega)$, and for a finite sequence of terms $\vec{t} = t_1, \ldots, t_n$, we define $\&\,\vec{t} = t_1\&(t_2\& \ldots (t_n\&\mathtt{t}) \ldots)$. If $t_1, \ldots, t_n$ are terms without fail, then $\&\,\vec{t} \longrightarrow^* \mathtt{t}$ if and only if $t_i \longrightarrow^* \mathtt{t}$ for all $i \leq n$, and thus the ordering of terms is not important. So the operation $\&\,A$ for a finite set A of terms of the sort $\mathbb{B}$ is well-defined provided that each $t \in A$ does not have the constant $\mathfrak{F}$. For a finite set $A = \{t_1, \ldots, t_k\}$ of terms of the sort $\mathbb{B}$, we define $\bigoplus A = t_1 \oplus (t_2 \oplus \ldots (t_k \oplus \mathtt{f}) \ldots)$. We have $\bigoplus A \longrightarrow^* t$ for $t \in A$ and $\bigoplus A \longrightarrow^* \mathtt{f}$.

The sort ϱ_d for representation of order-d stacks (and remainders) is defined by induction on $n - d$ by:

$$\varrho_n = \mathbb{B}$$
$$\varrho_d = \overbrace{\varrho_{d+1} \times \ldots \times \varrho_{d+1}}^{|Q|} \to \varrho_{d+1} \qquad (\text{if } d < n).$$

The sort environment for function symbols is given by:

$$\Delta = \{f_q^a :: \varrho_0 \mid q \in Q, a \in \Pi\}.$$

A configuration (q, α) with $\alpha = a_0 : \xi_1 : \xi_2 : \cdots : \xi_n$ is encoded as a term (with function symbols) of the form

$$f_q^{a_0}\, \langle \vec{v}_1 \rangle\, \langle \vec{v}_2 \rangle \ldots \langle \vec{v}_n \rangle,$$

where $\vec{v}_k$ is a sequence of length $|Q|$ consisting of values of sort ϱ_k, which represents the order-k remainder ξ_k. The precise description of the encoding will be presented later.

We define closed terms (containing function symbols) that simulate operations on higher-order stacks. Given $a \in \Pi$, $p \in Q$ and $op \in Op_n$, the closed term

$$[\![(a;\ q, op)]\!] :: \overbrace{\varrho_1 \times \ldots \times \varrho_1}^{|Q|} \times \cdots \times \overbrace{\varrho_n \times \ldots \times \varrho_n}^{|Q|} \to \mathbb{B}$$

is defined by:

$$[\![(a;\ q, pop_k)]\!] = \lambda\langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\rangle.\ x_{k,q}\,\langle \vec{x}_{k+1}\rangle \ldots \langle \vec{x}_n\rangle$$

$$[\![(a;\ q, push_1^b)]\!] = \lambda\langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\rangle.\ f_q^b \begin{bmatrix} f_1^a\langle \vec{x}_1\rangle \\ f_2^a\langle \vec{x}_1\rangle \\ \vdots \\ f_{|Q|}^a\langle \vec{x}_1\rangle \end{bmatrix} \langle \vec{x}_2\rangle \ldots \langle \vec{x}_n\rangle$$

$$[\![(a;\ q, push_k)]\!] = \lambda\langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\rangle.\ f_q^a\ \langle \vec{x}_1\rangle \ldots \langle \vec{x}_{k-1}\rangle \begin{bmatrix} f_1^a\langle \vec{x}_1\rangle \ldots \langle \vec{x}_k\rangle \\ f_2^a\langle \vec{x}_1\rangle \ldots \langle \vec{x}_k\rangle \\ \vdots \\ f_{|Q|}^a\langle \vec{x}_1\rangle \ldots \langle \vec{x}_k\rangle \end{bmatrix} \langle \vec{x}_{k+1}\rangle \ldots \langle \vec{x}_n\rangle$$

where

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix}$$

is an alternative notation for $\langle t_1, t_2, \ldots, t_m\rangle$.

We now define the program $P_{\mathcal{A}} = \texttt{let rec}\ D_{\mathcal{A}}\ \texttt{in}\ t_{\mathcal{A}}$. The function definition is given by:

$$D_{\mathcal{A}} = \{f_q^a = \lambda\langle \vec{x}_1\rangle.\lambda\langle \vec{x}_2\rangle.\ldots.\lambda\langle \vec{x}_n\rangle.t_q^a \mid q \in Q, a \in \Pi\}$$

where

$$t_q^a = \bigoplus_{\mathcal{O}\in\delta(q,a)} \&_{(p,op)\in\mathcal{O}}([\![(a;\ p, op)]\!]\langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\rangle).$$

(here $\mathcal{O} \in \delta(q, a)$ means $(q, a, \mathcal{O}) \in \delta$). For $k \leq n$, $\bot_k$ is defined as $\lambda\langle \vec{x}_{k+1}\rangle.\ldots.\lambda\langle \vec{x}_n\rangle.\texttt{f}$. For a pair of a state q and an order-k remainder ξ_k, $[\![(q, \xi_k)]\!]_k$ is a term of sort ϱ_k defined by:

$$[\![(q, \epsilon)]\!]_k = \bot_k$$

$$[\![(q, a{:}\xi_1{:}\ldots{:}\xi_k)]\!]_k = \lambda\langle \vec{x}_{k+1}\rangle.\ldots.\lambda\langle \vec{x}_n\rangle.f_q^a \begin{bmatrix} [\![(1, \xi_1)]\!]_1 \\ \vdots \\ [\![(|Q|, \xi_1)]\!]_1 \end{bmatrix} \ldots \begin{bmatrix} [\![(1, \xi_k)]\!]_k \\ \vdots \\ [\![(|Q|, \xi_k)]\!]_k \end{bmatrix} \langle \vec{x}_{k+1}\rangle \ldots \langle \vec{x}_n\rangle$$

The next lemma is a key to prove correctness of the reduction. The lemma shall be proved in the following subsection.

Lemma 32 (Correctness). *Let (q, α) be a configuration of a higher-order pushdown system $\mathcal{A}$. Then $(q, \alpha)\downarrow_{\mathcal{A}}$ if and only if $[\![(q, \alpha)]\!]_n \longrightarrow^*_{D_{\mathcal{A}}} \texttt{t}$.*

D.3 Correctness of the Simulation

Here we prove the correctness of the simulation (Lemma 32). We fix an order-n pushdown system $\mathcal{A}$.

A *witness of* $(q, \alpha)\downarrow$ is a derivation with the conclusion $(q, \alpha)\downarrow$, of which the inference rule is given by:

$$\frac{(q', op(\alpha))\downarrow \quad (\text{for all } (q', op) \in \mathcal{O})}{(q, \alpha)\downarrow}[(q, top(\alpha), \mathcal{O}) \in \delta].$$

Axioms are $(q, \alpha)\downarrow$ such that $(q, \alpha, \emptyset) \in \delta$. A witness ϖ is said to be *uniform* if it satisfies the following condition:

> If ϖ_1 and ϖ_2 are subderivations of the same conclusion, then ϖ_1 and ϖ_2 are identical.

Given a witness ϖ, one can construct a uniform witness ϖ' of the same conclusion. Hereafter, we shall consider only uniform witnesses. Note that every subderivation of a uniform witness is uniform. For a witness ϖ and a configuration (q, α), we write $(q, \alpha) \in \varpi$ if ϖ has a *proper* subderivation with the conclusion $(q, \alpha)\downarrow$.

Lemma 33. *Let $q \in Q$ be a state, $\alpha = a : \xi_1 : \cdots : \xi_n$ be a stack and ϖ be a witness of $(q, \alpha)\downarrow$. Assume that*

- *$v_{k,p} = [\![(p, \xi_k)]\!]_k$ for every $p \in Q$ and $k < n$, and*
- *$v_{n,p} \in \{\mathtt{t}, \mathtt{f}\}$ such that $(p, \xi_n) \in \varpi$ implies $v_{n,p} = \mathtt{t}$.*

Then $f_q^a \langle \vec{v}_1 \rangle \dots \langle \vec{v}_n \rangle \longrightarrow_{D_\mathcal{A}}^* \mathtt{t}$.

Proof. By induction on the structure of ϖ. Assume that the last rule used in ϖ is $(q, a, \mathcal{O}_1) \in \delta$. Then by definition of $D_\mathcal{A}$,

$$\begin{aligned} f_q^a \langle \vec{v}_1 \rangle \dots \langle \vec{v}_n \rangle &\longrightarrow_{D_\mathcal{A}}^* \bigoplus_{\mathcal{O} \in \delta(q,a)} \&_{(p,op)\in\mathcal{O}} ([\![(a; p, op)]\!] \langle \vec{v}_1, \vec{v}_2, \dots, \vec{v}_n \rangle) \\ &\longrightarrow_{D_\mathcal{A}}^* \&_{(p,op)\in\mathcal{O}_1} ([\![(a; p, op)]\!] \, \langle \vec{v}_1, \vec{v}_2, \dots, \vec{v}_n \rangle). \end{aligned}$$

It suffices to show that $[\![(a; p, op)]\!] \, \langle \vec{v}_1, \dots, \vec{v}_n \rangle \longrightarrow_{D_\mathcal{A}}^* \mathtt{t}$ for every $(p, op) \in \mathcal{O}_1$. We do case analysis on op. Note that, for every $(p, op) \in \mathcal{O}_1$, we have a witness ϖ' of $(p, op(\alpha))$ that is a subderivation of ϖ.

Case $op = pop_k$ for some $k < n$: Then ξ_k is not empty since $(p, pop_k(\alpha))\downarrow$. Assume that $\xi_k = a' : \xi_1' : \cdots : \xi_k'$. By the definition of $[\![(a; q, pop_k)]\!]$, we have

$$[\![(a; p, op)]\!] \, \langle \vec{v}_1, \dots, \vec{v}_n \rangle \longrightarrow v_{k,p} \, \langle \vec{v}_{k+1} \rangle \dots \langle \vec{v}_n \rangle.$$

By the assumption, we have $v_{k,p} = [\![(p, \xi_k)]\!]_k$ and thus

$$v_{k,p} = \lambda \langle \vec{x}_{k+1} \rangle . \dots . \lambda \langle \vec{x}_n \rangle . f_p^{a'} \, \langle \vec{v}_1' \rangle \dots \langle \vec{v}_k' \rangle \langle \vec{x}_{k+1} \rangle \dots \langle \vec{x}_n \rangle,$$

where $v'_{i,p'} = [\![(p', \xi'_i)]\!]_i$ for every $i \in \{1, \ldots, k\}$ and $p' \in Q$. Therefore

$$v_{k,p} \langle \vec{v}_{k+1} \rangle \ldots \langle \vec{v}_n \rangle \longrightarrow^* f_p^{a'} \langle \vec{v}'_1 \rangle \ldots \langle \vec{v}'_k \rangle \langle \vec{v}_{k+1} \rangle \ldots \langle v_n \rangle \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}.$$

Here we use the induction hypothesis for the second step (note that $(p, pop_k(\alpha)) = \xi_k : \xi_{k+1} : \cdots : \xi_n = a' : \xi'_1 : \cdots : \xi'_k : \xi_{k+1} : \cdots : \xi_n$).

Case $op = pop_n$: Then by the definition of $[\![(a; p, op)]\!]$, we have

$$[\![(a; p, op)]\!] \langle \vec{v}_1, \ldots, \vec{v}_n \rangle \longrightarrow v_{n,p}.$$

Since $(p, \xi_n) = (p, pop_n(\alpha)) \in \varpi$, we have $v_{n,p} = \mathtt{t}$ by the assumption.

Case $op = push_n$: Then by the definition of $[\![(a; p, op)]\!]$, we have

$$[\![(a; p, op)]\!] \langle \vec{v}_1, \ldots, \vec{v}_n \rangle \longrightarrow f_p^a \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_{n-1} \rangle \langle \vec{u} \rangle,$$

where $u_{p'} = f_{p'}^a \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_n \rangle$ for every $p' \in Q$. By the definition of $f_{p'}^a$ in $D_\mathcal{A}$, we have

$$u_{p'} \longrightarrow^*_{D_\mathcal{A}} \mathtt{f}$$

for every $p' \in Q$. Furthermore, by the induction hypothesis, if $(p', \alpha) \in \varpi$, then we have

$$u_{p'} \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}.$$

We define $w_{p'} = \mathtt{t}$ if $(p', \alpha) \in \varpi$ and $w_{p'} = \mathtt{f}$ otherwise. Then $u_{p'} \longrightarrow^*_{D_\mathcal{A}} w_{p'}$ for every $p' \in Q$. Let ϖ' be the subderivation of ϖ whose conclusion is $(p, push_n(\alpha))\downarrow$. Then $(p', \alpha) \in \varpi'$ implies $w_{p'} = \mathtt{t}$. Thus

$$\begin{aligned} f_p^a \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_{n-1} \rangle \langle \vec{u} \rangle &\longrightarrow_{D_\mathcal{A}} (\lambda \langle \vec{x}_1 \rangle \ldots \lambda \langle \vec{x}_n \rangle . t_p^a) \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_{n-1} \rangle \langle \vec{u} \rangle \\ &\longrightarrow^* \left(\lambda \langle \vec{x}_n \rangle . ([\vec{v}_1/\vec{x}_1, \ldots, \vec{v}_{n-1}/\vec{x}_{n-1}] t_p^a)\right) \langle \vec{u} \rangle \\ &\longrightarrow^*_{D_\mathcal{A}} \left(\lambda \langle \vec{x}_n \rangle . ([\vec{v}_1/\vec{x}_1, \ldots, \vec{v}_{n-1}/\vec{x}_{n-1}] t_p^a)\right) \langle \vec{w} \rangle \\ &\longrightarrow^*_{D_\mathcal{A}} \mathtt{t}. \end{aligned}$$

Here we use the induction hypothesis for the last step.

Other cases can be proved similarly. □

Lemma 34. *Let $q \in Q$ be a state and $\alpha = a : \xi_1 : \cdots : \xi_n$ be a stack. Assume that*

- *$v_{k,p} = [\![(p, \xi_k)]\!]_k$ for every $p \in Q$ and $k < n$, and*
- *$v_{n,p} \in \{\mathtt{t}, \mathtt{f}\}$ such that $v_{n,p} = \mathtt{t}$ implies that $(p, \xi_n)\downarrow$.*

If $f_q^a \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_n \rangle \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}$, then $(q, \alpha)\downarrow$.

Proof. By induction on the length of the reduction sequence. By the definition of f_q^a in $D_\mathcal{A}$, the reduction sequence must be of the form

$$\begin{aligned} f_q^a \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_n \rangle &\longrightarrow^*_{D_\mathcal{A}} \bigoplus_{\mathcal{O} \in \delta(q,a)} \&_{(p,op) \in \mathcal{O}} ([\![(a; p, op)]\!] \langle \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \rangle) \\ &\longrightarrow^*_{D_\mathcal{A}} \&_{(p,op) \in \mathcal{O}_1} ([\![(a; p, op)]\!] \langle \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \rangle) \\ &\longrightarrow^*_{D_\mathcal{A}} \mathtt{t} \end{aligned}$$

for some $\mathcal{O}_1 \in \delta(q,a)$. If $\mathcal{O}_1$ is the empty set, i.e. $(q,a,\emptyset) \in \delta$, then $(q, a : \xi_1 : \cdots : \xi_n)\downarrow$ as desired. Assume that $\mathcal{O}_1$ is not empty.

By the definition of &, we know that

$$[\![(a;p,op)]\!]\,\langle \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\rangle \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}$$

for every $(p,op) \in \mathcal{O}_1$. Let $(p,op) \in \mathcal{O}_1$. Assume that $op(\alpha) = a' : \xi'_1 : \cdots : \xi'_n$. By an argument similar to the proof of Lemma 33, this reduction sequence must be of the form

$$[\![(a;p,op)]\!]\,\langle \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\rangle \longrightarrow^*_{D_\mathcal{A}} f_p^{a'}\,\langle \vec{v}'_1\rangle \ldots \langle \vec{v}'_n\rangle,$$

where

- $v'_{i,p'} = [\![(p', \xi'_i)]\!]_i$ for every $i \in \{1, \ldots, n-1\}$ and $p' \in Q$, and
- $v'_{n,p'} = \{\mathtt{t}, \mathtt{f}\}$ such that $v'_{n,p'} = \mathtt{t}$ implies $(p', \xi'_n)\downarrow$ (here we use the induction hypothesis to prove this claim).

So by the induction hypothesis, we have $(p, a' : \xi'_1 : \cdots : \xi'_n)\downarrow$, which is equivalent to $(q, op(\alpha))\downarrow$. □

Lemma 35. *Let $q \in Q$ and ξ_n be an order-n remainder. Then $[\![(q,\xi_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}}$ $\mathtt{f}$.*

Proof. By induction on ξ_n. If $\xi_n = \varepsilon$, then $[\![(q,\xi_n)]\!]_n = \mathtt{f}$. Assume that $\xi_n = a : \xi'_1 : \cdots : \xi'_n$. Then we have

$$[\![(q,\xi_n)]\!]_n = f_q^a\,\langle v_1\rangle \ldots \langle v_{n-1}\rangle \begin{bmatrix} [\![(1,\xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|,\xi'_n)]\!]_n \end{bmatrix},$$

where $v_{k,p} = [\![(p,\xi'_k)]\!]_k$ for every $k < n$ and $p \in Q$. By definition, $v_{k,p}$ is a value. By the induction hypothesis, $[\![(p,\xi'_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}} \mathtt{f}$. Therefore

$$\begin{aligned}
[\![(q,\xi_n)]\!]_n &\longrightarrow_{D_\mathcal{A}} (\lambda\langle \vec{x}_1\rangle.\ldots.\lambda\langle \vec{x}_n\rangle.t_q^a)\,\langle \vec{v}_1\rangle \ldots \langle \vec{v}_{n-1}\rangle \begin{bmatrix} [\![(1,\xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|,\xi'_n)]\!]_n \end{bmatrix} \\
&\longrightarrow^* \left(\lambda\langle \vec{x}_n\rangle.([\vec{v}_1/\vec{x}_1, \ldots, \vec{v}_{n-1}/\vec{x}_{n-1}]t_q^a)\right) \begin{bmatrix} [\![(1,\xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|,\xi'_n)]\!]_n \end{bmatrix} \\
&\longrightarrow^*_{D_\mathcal{A}} \left(\lambda\langle \vec{x}_n\rangle.([\vec{v}_1/\vec{x}_1, \ldots, \vec{v}_{n-1}/\vec{x}_{n-1}]t_q^a)\right)\langle \vec{\mathtt{f}}\rangle \\
&\longrightarrow_{D_\mathcal{A}} \bigoplus_{\mathcal{O}\in\delta(q,a)} \&_{(p,op)\in\mathcal{O}}([\![(a;\ p,op)]\!]\langle \vec{v}_1, \ldots, \vec{v}_{n-1}, \vec{\mathtt{f}}\rangle) \\
&\longrightarrow^* \mathtt{f}.
\end{aligned}$$

□

Proof (Lemma 32). By induction on ξ_n. If $\xi_n = \varepsilon$, then $(q, \xi_n)\downarrow$ never holds and $[\![(q, \xi_n)]\!]_n = \mathtt{f}$. Assume that $\xi_n = a' : \xi'_1 : \cdots : \xi'_n$.

We prove the left-to-right direction. Assume that $(q, a' : \xi'_1 : \cdots : \xi'_n)\downarrow$. Then we have a witness ϖ of $(q, a' : \xi'_1 : \cdots : \xi'_n)\downarrow$. For every $p \in Q$, we define $v_p = \mathtt{t}$ if $(p, \xi'_n) \in \varpi$ and $v_p = \mathtt{f}$ otherwise. By Lemma 35 and the induction hypothesis, we have $[\![(p, \xi'_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}} v_p$. So by Lemma 33, we have $[\![(q, \xi_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}$.

We prove the right-to-left direction. Assume that $[\![(q, a' : \xi'_1 : \cdots : \xi'_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}$. Let $v_{k,p} = [\![(p, \xi'_k)]\!]_k$ for every $k \in \{1, \ldots, n-1\}$ and $p \in Q$. Then the reduction sequence must be of the form

$$
\begin{aligned}
&[\![(q, a' : \xi'_1 : \ldots : \xi'_n)]\!]_n \\
&= f_q^{a'} \, \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_{n-1} \rangle \begin{bmatrix} [\![(1, \xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|, \xi'_n)]\!]_n \end{bmatrix} \\
&\longrightarrow_{D_\mathcal{A}} (\lambda \langle \vec{x}_1 \rangle . \ldots . \lambda \langle \vec{x}_n \rangle . t_q^{a'}) \, \langle \vec{v}_1 \rangle \ldots \langle \vec{v}_{n-1} \rangle \begin{bmatrix} [\![(1, \xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|, \xi'_n)]\!]_n \end{bmatrix} \\
&\longrightarrow^* \left(\lambda \langle \vec{x}_n \rangle . ([\vec{v}_1 / \vec{x}_1, \ldots, \vec{v}_{n-1} / \vec{x}_{n-1}] t_q^{a'}) \right) \begin{bmatrix} [\![(1, \xi'_n)]\!]_n \\ \vdots \\ [\![(|Q|, \xi'_n)]\!]_n \end{bmatrix} \\
&\longrightarrow^*_{D_\mathcal{A}} \left(\lambda \langle \vec{x}_n \rangle . ([\vec{v}_1 / \vec{x}_1, \ldots, \vec{v}_{n-1} / \vec{x}_{n-1}] t_q^{a'}) \right) \langle \vec{w} \rangle \\
&\longrightarrow^*_{D_\mathcal{A}} \mathtt{t}.
\end{aligned}
$$

Here $[\![(p, \xi'_n)]\!]_n \longrightarrow^*_{D_\mathcal{A}} w_p \in \{\mathtt{t}, \mathtt{f}\}$ for every p. By the induction hypothesis, $v_p = \mathtt{t}$ implies $(p, \xi'_n)\downarrow$. So by Lemma 34, we have $(q, a' : \xi'_1 : \cdots : \xi'_n)\downarrow$. □

Proof (Theorem 2). A consequence of Lemma 31 and Lemma 32. The program corresponding to $(q, \alpha)\downarrow_\mathcal{A}$ is $\mathtt{let\ rec}\ D_\mathcal{A}\ \mathtt{in}\ \mathbf{if}([\![(q, \alpha)]\!]_n, \mathfrak{F}, \Omega)$. Note that $D_\mathcal{A}$ and $[\![(q, \alpha)]\!]_n$ do not contain $\mathfrak{F}$, and thus $\mathtt{let\ rec}\ D_\mathcal{A}\ \mathtt{in}\ \mathbf{if}([\![(q, \alpha)]\!]_n, \mathfrak{F}, \Omega)$ fails if and only if $[\![(q, \alpha)]\!]_n \longrightarrow^*_{D_\mathcal{A}} \mathtt{t}$. □

E Proof of Soundness and Completeness (Theorem 3)

We first prove some useful lemmas.

If $\Gamma :: \mathcal{K}$ and $\Gamma' :: \mathcal{K}$, we define $\Gamma \wedge \Gamma'$ by $(\Gamma \wedge \Gamma')(x) = \Gamma(x) \wedge \Gamma'(x)$.

Lemma 36 (Weakening). *If $\Gamma \vdash t : \tau$, then $\Gamma \wedge \Gamma' \vdash t : \tau$ for every Γ' (provided that $\Gamma \wedge \Gamma'$ is defined).*

Proof. By induction on the structure of the derivation of $\Gamma \vdash t : \tau$. □

The intersection introduction rule is admissible for values.

Lemma 37 (Intersection). *If $\Gamma \vdash v : \vartheta_1$ and $\Gamma \vdash v : \vartheta_2$, then $\Gamma \vdash v : \vartheta_1 \wedge \vartheta_2$.*

Proof. By the case analysis of v. If $v = \mathtt{t}$, then $\vartheta_1 = \vartheta_2 = \mathtt{t}$ and thus $\vartheta_1 \wedge \vartheta_2 = \mathtt{t}$. If $v = \mathtt{f}$, then $\vartheta_1 = \vartheta_2 = \mathtt{f}$ and thus $\vartheta_1 \wedge \vartheta_2 = \mathtt{f}$. Assume that $v = \lambda\langle\vec{x}\rangle.t$. Then ϑ_1 and ϑ_2 must be of the form

$$\vartheta_1 = \bigwedge_{i \in I} (\vec{\vartheta}'_i \to \tau_i) \quad \text{and} \quad \vartheta_2 = \bigwedge_{j \in J} (\vec{\vartheta}'_j \to \tau_j).$$

It suffices to prove that $\Gamma \vdash \lambda\langle\vec{x}\rangle.t : \bigwedge_{i \in I \cup J} (\vec{\vartheta}'_i \to \tau_i)$. To this end, one needs to prove

$$\Gamma, \vec{x} : \vec{\vartheta}'_i \vdash t : \tau_i$$

for every $i \in I \cup J$. If $i \in I$, then a derivation of the above judgement appears in the derivation of $\Gamma \vdash \lambda\langle\vec{x}\rangle.t : \bigwedge_{i \in I} (\vec{\vartheta}'_i \to \tau_i)$. Similarly for $i \in J$. □

E.1 Soundness

Soundness of the type system can be proved straightforwardly, using Substitution Lemma (Lemma 39) and Progress Lemma (Lemma 40).

Lemma 38. *If $\Gamma \vdash v : \tau \wedge \tau'$, then $\Gamma \vdash v : \tau$.*

Proof. Recall that $v = \mathtt{t}$ or $\mathtt{f}$ or $\lambda\langle\vec{x}\rangle.t$. If $v = \mathtt{t}$, then $\tau \wedge \tau' = \mathtt{t}$ and thus $\tau = \tau' = \mathtt{t}$. The case that $v = \mathtt{f}$ can be proved similarly.

Assume $v = \lambda\langle\vec{x}\rangle.t$. Then $\tau = \bigwedge_{i \in I} (\vec{\vartheta}_i \to \psi_i)$ and $\tau' = \bigwedge_{i \in I'} (\vec{\vartheta}_j \to \psi_j)$ for some I and I', and $\tau \wedge \sigma = \bigwedge_{i \in I \cup I'} (\vec{\vartheta}_i \to \psi_i)$. Since $\Gamma \vdash \lambda\langle\vec{x}\rangle.t : \tau \wedge \sigma$, we have $\Gamma, \vec{x} : \vec{\vartheta}_i \vdash t : \psi_i$ for every $i \in I \cup I'$. Therefore $\Gamma, \vec{x} : \vec{\vartheta}_i \vdash t : \psi_i$ for every $i \in I$. By (Abs) rule, we have $\Gamma \vdash \lambda\langle\vec{x}\rangle.t : \bigwedge_{i \in I} (\vec{\vartheta}_i \to \psi_i)$. □

Lemma 39 (Substitution). *If $\Gamma, x : \tau \vdash t : \sigma$ and $\Gamma \vdash v : \tau$, then $\Gamma \vdash [v/x]t : \sigma$.*

Proof. By induction on the structure of t. The proof of the case $t = x$ needs Lemma 38. □

Lemma 40 (Progress). *Suppose that $\vdash t : \tau$. Then*

- *$t \longrightarrow t'$ and $\vdash t' : \tau$ for some t',*
- *$\tau = \mathfrak{F}$ and $t = E[\mathfrak{F}]$, or*
- *$\tau \neq \mathfrak{F}$ and t is a value.*

Proof. By induction on the derivation of $\vdash t : \tau$. We do case analysis on the last rule used in $\vdash t : \tau$.

- Case ($\mathfrak{F}$): Then $t = \mathfrak{F}$ and condition (2) holds.
- Case (App): Then $t = s\,\langle\vec{u}\rangle$ and $\vdash s : \bigwedge_{i \in I} (\vec{\sigma}_i \to \vartheta_i)$ and there exists $l \in I$ such that $\vdash u_k : \sigma_{k,l}$ (for every $k \in \{1, \ldots, n\}$) and $\tau = \vartheta_l$. By the induction hypothesis, $s \longrightarrow s'$ and $\vdash s' : \bigwedge_{i \in I} (\vec{\sigma}_i \to \vartheta_i)$ or s is a value. For the former case, we have $\vdash s'\,\langle\vec{u}\rangle : \tau$. Assume the latter case. Then we check if u_k is not

a value for some k. If so, by the induction hypothesis, we have $u_k \longrightarrow u'_k$ and this gives a reduction

$$s\,\langle u_1, \ldots, u_{k-1}, u_k, u_{k+1}, \ldots, u_n\rangle \longrightarrow s\,\langle u_1, \ldots, u_{k-1}, u'_k, u_{k+1}, \ldots u_n\rangle$$

that preserves typing. Assume that s and u_k (for all $k \in [1, n]$) are values. Then $s = \lambda\langle\vec{x}\rangle.s'$ and $\vec{x} : \vec{\sigma}_i \vdash s' : \vartheta_i$ for every $i \in I$. Since $s\,\langle\vec{u}\rangle \longrightarrow [\vec{u}/\vec{x}]s'$, it suffices to show that $\vdash [\vec{u}/\vec{x}]s' : \vartheta_l$, which is a consequence of Substitution Lemma (Lemma 39).

- Case (App-$\mathfrak{F}$1): Then $t = s\,\langle\vec{u}\rangle$ and $\vdash s : \mathfrak{F}$. By the induction hypothesis, we have s' such that $s \longrightarrow s'$ and $\vdash s' : \mathfrak{F}$ or $s = E[\mathfrak{F}]$ for some evaluation context E. In the former case, we have $s\,\langle\vec{u}\rangle \longrightarrow s'\,\langle\vec{u}\rangle$ and $\vdash s'\,\langle\vec{u}\rangle : \mathfrak{F}$. In the latter case, $t = E'[\mathfrak{F}]$, where $E' = E\,\langle\vec{u}\rangle$.
- Case (Br-1): Then $t = s_1 \oplus s_2$ and $\vdash s_1 : \tau$. We have $s_1 \oplus s_2 \longrightarrow s_1$ as desired.

Other cases can be proved easily. □

Theorem 5 (Soundness). *Let $P = \texttt{let rec}\ D\ \texttt{in}\ t$ be a program. If $\vdash P : \mathfrak{F}$, then $t \longrightarrow^*_D E[\mathfrak{F}]$.*

Proof. Assume $\vdash_D t : \mathfrak{F}$, i.e. $\vdash [t]^n_D : \mathfrak{F}$ for some n. By iteratively applying Lemma 40, we have either a reduction sequence of the form

$$[t]^n_D \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_i \longrightarrow E[\mathfrak{F}].$$

or an infinite reduction sequence, but the latter is not possible by Lemma 1. By Lemma 2, we have $t \longrightarrow^*_D E'[\mathfrak{F}]$ for some E'. □

E.2 Completeness

A key to prove completeness is the admissibility of the intersection introduction rule for values (Lemma 37). Using this lemma, we have De-Substitution Lemma for *substitution of values* and Subject Expansion for *call-by-value reduction*.

Lemma 41 (De-Substitution). *Assume that $\Gamma \vdash [\vec{v}/\vec{x}]t : \tau$. Then there exists $\vec{\theta}$ such that $\Gamma, \vec{x} : \vec{\theta} \vdash t : \tau$ and $\Gamma \vdash \vec{v} : \vec{\theta}$.*

Proof. By induction on the structure of t, using Lemma 37. □

Lemma 42. *If $\vdash E[t] : \tau$, then $x : \theta \vdash E[x] : \tau$ and $\vdash t : \theta$ for some θ. Conversely, $x : \theta \vdash E[x] : \tau$ and $\vdash t : \theta$ implies $\vdash E[t] : \theta$.*

Proof. By induction on the structure of E. □

Lemma 43 (Subject Expansion). *If $\vdash t' : \tau$ and $t \longrightarrow t'$, then $\vdash t : \tau$.*

Proof. Assume that $t \longrightarrow t'$ and $\vdash t' : \tau$. We prove the lemma by case analysis on the reduction rules:

– Case $E[(\lambda\langle\vec{x}\rangle.t)\,\langle\vec{v}\rangle] \longrightarrow E[\,[\vec{v}/\vec{x}]t]$: By Lemma 42, $y : \sigma \vdash E[y] : \tau$ and $\vdash [\vec{v}/\vec{x}]t : \sigma$ for some σ. By De-Substitution Lemma (Lemma 41), there exists $\vec{\vartheta}$ such that $\vec{x} : \vec{\vartheta} \vdash t : \sigma$ and $\vdash \vec{v} : \vec{\vartheta}$. By applying (Abs) rule, we have $\vdash \lambda\langle\vec{x}\rangle.t : \bigwedge\{\vec{\vartheta} \to \sigma\}$. By (App) rule, $\vdash (\lambda\langle\vec{x}\rangle.t)\,\langle\vec{v}\rangle : \sigma$ and thus $\vdash E[(\lambda\langle\vec{x}\rangle.t)\,\langle\vec{v}\rangle] : \tau$ by Lemma 42.
– Case $E[t_1 \oplus t_2] \longrightarrow E[t_1]$: By Lemma 42, $x : \sigma \vdash E[x] : \tau$ and $\vdash t_1 : \sigma$ for some σ. By (Br-1) rule, we have $\vdash t_1 \oplus t_2 : \sigma$. Thus $\vdash E[t_1 \oplus t_2] : \tau$.
– Case $E[\mathbf{if}(\mathtt{t}, t_1, t_2)] \longrightarrow E[t_1]$: By Lemma 42, $x : \sigma \vdash E[x] : \tau$ and $\vdash t_1 : \sigma$ for some σ. Then it is easy to prove $\vdash \mathbf{if}(\mathtt{t}, t_1, t_2) : \sigma$ and thus $\vdash E[\mathbf{if}(\mathtt{t}, t_1, t_2)] : \tau$ by Lemma 42.

Other cases can be proved similarly. □

Theorem 6 (Completeness). *Let* $P = \mathtt{let\,rec}\ D\ \mathtt{in}\ t$ *be a program. If* $t \longrightarrow_D^* E[\mathfrak{F}]$*, then* $\vdash P : \mathfrak{F}$*.*

Proof. Assume $t \longrightarrow_D^* E[\mathfrak{F}]$. Then by Lemma 2, $[t]_D^n \longrightarrow^* E'[\mathfrak{F}]$ for some n. Using Subject Expansion (Lemma 43), $\vdash [t]_D^n : \mathfrak{F}$ can be proved by induction on the length of the reduction sequence. Thus $\vdash P : \mathfrak{F}$. □

Proof (Theorem 3). A consequence of Soundness (Theorem 5) and Completeness (Theorem 6). □

F Proof of Lemma 6

Assume that $\Delta = \{f_1 :: \delta_1, \ldots, f_k :: \delta_k\}$ and let F_i^m be the mth approximation of f_i (see Section 2 for the definition of F_i^m).

Lemma 44. *For every function symbol* f_i*,* $\vdash F_i^m : \tau$ *if and only if* $\tau \succeq \Theta_m(f_i)$*.*

Proof. By induction on m. The base case is trivial. We prove the induction step.

For the left-to-right direction, suppose that $\vdash F_i^{m+1} : \tau$. By definition of F_i^{m+1}, we have

$$\vdash [F_1^m/f_1, \ldots, F_k^m/f_k](D(f_i)) : \tau.$$

By De-substitution Lemma (Lemma 41), we have $f_1 : \theta_1, \ldots, f_k : \theta_k \vdash D(f_i) : \tau$ and $\vdash F_j^m : \theta_j$ for every $j \le k$ (note that F_j^m is a value for every m and j). By the induction hypothesis, $\theta_j \succeq \Theta_m(f_j)$ for every $j \le k$. So by Lemma 36, we have $\Theta_m \vdash D(f_i) : \tau$. Hence $\tau \succeq \Theta_{m+1}(f_i)$ by definition of Θ_{m+1}.

For the right-to-left direction, suppose that $\tau \succeq \Theta_{m+1}(f_i)$. Then by definition of $\Theta_{m+1}(f_i)$, we have $\Theta_m \vdash D(f_i) : \tau \wedge \tau'$ for some τ'. Since $D(f_i)$ is a value, we know that $\Gamma_m^D \vdash D(f_i) : \tau$ by Lemma 38. By the induction hypothesis, for every $j \le k$, we have $\vdash F_j^m : \Theta_m(f_j)$. So by Substitution Lemma (Lemma 39), $\vdash [F_1^m/f_1, \ldots, F_k^m/f_k]D(f_i) : \tau$ as desired. □

Lemma 6 is a consequence of Lemma 44, De-Substitution Lemma (Lemma 41) and Substitution Lemma (Lemma 39).

G Proof of Lemma 7

Lemma 45. *Let $n, m_1, \ldots, m_k$ be positive natural numbers (i.e. $n \geq 1$ and $m_i \geq 1$ for every i). Then*

$$\prod_{i=0}^{k} \mathbf{exp}_n(m_i) \leq \mathbf{exp}_n(\sum_{i=0}^{k} m_i)\,.$$

Proof. By induction on n. If $n = 1$, then

$$\prod_{i=0}^{k} \mathbf{exp}_1(m_i) = \prod_{i=0}^{k} 2^{m_i} = 2^{\sum_{i=0}^{k} m_i}.$$

Assume that $n \geq 2$. Note that $\mathbf{exp}_{n-1}(m_i) \geq 2$. Thus $\sum_{i=0}^{k} \mathbf{exp}_{n-1}(m_i) \leq \prod_{i=0}^{k} \mathbf{exp}_{n-1}(m_i)$. So, by using the induction hypothesis, we have

$$\begin{aligned}
\prod_{i=0}^{k} \mathbf{exp}_n(m_i) &= \prod_{i=0}^{k} 2^{\mathbf{exp}_{n-1}(m_i)} \\
&= 2^{\sum_{i=0}^{k} \mathbf{exp}_{n-1}(m_i)} \\
&\leq 2^{\prod_{i=0}^{k} \mathbf{exp}_{n-1}(m_i)} \\
&\leq 2^{\mathbf{exp}_{n-1}(\sum_{i=0}^{k} m_i)} \\
&= \mathbf{exp}_n(\sum_{i=0}^{k} m_i)
\end{aligned}$$

as desired. □

Proof (Proof of Lemma 7). By induction on the structure of κ. If $\kappa = \mathbb{B}$, then $\mathcal{T}(\mathbb{B}) = \{\mathtt{t}, \mathtt{f}, \mathfrak{F}_{\mathbb{B}}\}$ and $n = 0$ and $|\mathbb{B}| = 1$. It is easy to see that $\#\mathcal{T}(\mathbb{B}) = 3 \leq 2^2 = \mathbf{exp}_{n+1}(2|\mathbb{B}|)$. The height of $\mathcal{T}(\kappa)$ is 1, which is less than $\mathbf{exp}_0(2|\mathbb{B}|) = 2$.

Assume that $\kappa = \kappa_1 \times \ldots \times \kappa_k \to \iota$. Then

$$\mathcal{T}(\kappa_1 \times \ldots \times \kappa_k \to \iota) \cong \mathcal{P}\Big((\mathcal{T}(\kappa_1) - \{\mathfrak{F}\}) \times \cdots \times (\mathcal{T}(\kappa_k) - \{\mathfrak{F}\}) \times \mathcal{T}(\iota)\Big),$$

where $\mathcal{P}(A)$ is the set of all subsets of A. So $\#\mathcal{T}(\kappa)$ can be estimated, using the induction hypothesis and Lemma 45, as:

$$\begin{aligned}
\#\mathcal{T}(\kappa_1 \times \ldots \times \kappa_k \to \iota) &= \#\mathcal{P}\Big((\mathcal{T}(\kappa_1) - \{\mathfrak{F}\}) \times \ldots (\mathcal{T}(\kappa_k) - \{\mathfrak{F}\}) \times \mathcal{T}(\iota)\Big) \\
&= 2^{(\#\mathcal{T}(\kappa_1)-1) \times \cdots \times (\#\mathcal{T}(\kappa_k)-1) \times \#\mathcal{T}(\iota)} \\
&\leq 2^{\mathbf{exp}_n(2|\kappa_1|) \times \cdots \times \mathbf{exp}_n(2|\kappa_k|) \times \mathbf{exp}_n(2|\iota|)} \\
&\leq 2^{\mathbf{exp}_n(2(|\kappa_1| + \cdots + |\kappa_k| + |\iota|))} \\
&< 2^{\mathbf{exp}_n(2|\kappa|)} \\
&= \mathbf{exp}_{n+1}(2|\kappa|)\,.
\end{aligned}$$

It is easy to see that the height of $\mathcal{T}(\kappa)$ is bounded by the number of elements in $\mathcal{T}(\kappa_1) \times \cdots \times \mathcal{T}(\kappa_k) \times \mathcal{T}(\iota)$, which is bounded by $\mathbf{exp}_n(2|\kappa|)$. □

H Proof of Claim in Proof of Lemma 8

For a term t with $\Delta \mid \mathcal{K} \vdash t :: \kappa$ and $\Theta :: \Delta$, we define $A_{\Theta,t} \subseteq \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa)$ by

$$A_{\Theta,t} = \{(\Gamma, \tau) \in \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa) \mid \Delta, \Gamma \vdash t : \tau\}.$$

Lemma 8 states that $A_{\Theta,t}$ can be computed in time $O(\mathbf{exp}_n(\mathit{poly}(|t|))$ under certain conditions, where $n \geq \max\{\mathit{depth}(t), \mathit{depth}(\mathcal{K})\}$.

For a subset $A \subseteq \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa)$ and $\Gamma \in \mathcal{T}(\mathcal{K})$, we define $A \restriction \Gamma$ by

$$(A \restriction \Gamma) = \{\tau \mid (\Gamma, \tau) \in A\}.$$

We prove the following claim used in the proof of Lemma 8.

Lemma 46. *Assume that $\mathit{depth}(t) = \mathit{depth}(\kappa) = n$ and $\mathit{depth}(\mathcal{K}) < n$. Then there exists $B_{\Theta,t} \subseteq \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa)$ that satisfies the following properties.*

1. *$(\Gamma, \tau) \in A_{\Theta,t}$ if and only if $(\Gamma, \tau') \in B_{\Theta,t}$ for some $\tau' \preceq \tau$.*
2. *$\#(B_{\Theta,t} \restriction \Gamma) \leq |t|$ for every Γ.*

Proof. By induction on the structure of t.

Case $t = f$ for some function symbol f: Take $B_{\Theta,f} = \{(\Gamma, \Theta(f)) \mid \Gamma \in \mathcal{T}(\mathcal{K})\}$.

Case $t = \lambda\langle\vec{x}\rangle.u$: For every $\Gamma :: \mathcal{K}$, we define

$$\tau_\Gamma = \bigwedge\{\vec{\theta} \to \sigma \mid \Theta, \Gamma, \vec{x} : \vec{\theta} \vdash u : \sigma\}.$$

It is easy to see that $\Theta, \Gamma \vdash \lambda\langle\vec{x}\rangle.u : \tau_0$ implies $\tau_0 \succeq \tau_\Gamma$. Recall that

$$\bigwedge_{i \in I}(\vec{\theta}_i \to \sigma_i) \succeq \bigwedge_{j \in J}(\vec{\theta}_j \to \sigma_j)$$

intuitively means that $I \subseteq J$. So take $B_{\Theta,\lambda\langle\vec{x}\rangle.u} = \{(\Gamma, \tau_\Gamma) \mid \Gamma :: \mathcal{K}\}$.

Case $t = t_1 \oplus t_2$: Take $B_{\Theta,t_1 \oplus t_2} = B_{\Theta,t_1} \cup B_{\Theta,t_2}$. Then $B_{\Theta,t_1 \oplus t_2}$ satisfies the condition (2). We prove the condition (1). Assume that $(\Gamma, \tau) \in A_{\Theta,t_1 \oplus t_2}$. Then $\Theta, \Gamma \vdash t_1 \oplus t_2 : \tau$. Thus $\Theta, \Gamma \vdash t_i : \tau$ for some $i \in \{1, 2\}$, which means that $(\Gamma, \tau) \in A_{\Theta,t_i}$. By the induction hypothesis, we have $(\Gamma, \tau') \in B_{\Theta,t_i}$ for some $\tau' \preceq \tau$. Then $(\Gamma, \tau') \in B_{\Theta,t_1 \oplus t_2}$ as expected. The converse can be proved similarly.

Case $t = \mathbf{if}(s, u_1, u_2)$: We define $B_{\Theta,\mathbf{if}(s,u_1,u_2)}$ by

$$B_{\Theta,\mathbf{if}(s,u_1,u_2)} = \{(\Gamma, \tau) \in B_{\Theta,u_1} \mid (\Gamma, \mathtt{t}) \in A_{\Theta,s}\} \cup \{(\Gamma, \tau) \in B_{\Theta,u_2} \mid (\Gamma, \mathtt{f}) \in A_{\Theta,s}\}.$$

It is easy to check that $B_{\Theta,\mathbf{if}(s,u_1,u_2)}$ meets the conditions.

Case $t = s\,\langle\vec{u}\rangle$: This contradicts the assumption that $\mathit{depth}(t) = \mathit{depth}(\kappa)$, since the sort of s is $\vec{\iota} \to \kappa$ for some $\vec{\iota}$ and $\mathit{depth}(\vec{\iota} \to \kappa) > \mathit{depth}(\kappa)$. □