Logical Methods in Computer Science
Volume 21, Issue 3, 2025, pp. 17:1–17:40
https://lmcs.episciences.org/

Submitted Jul. 03, 2023
Published Aug. 07, 2025

A FULLY ABSTRACT MODEL OF PCF BASED ON EXTENDED ADDRESSING MACHINES

BENEDETTO INTRIGILA [a], GIULIO MANZONETTO [b], AND NICOLAS MÜNNICH [c]

[a] Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Italy
e-mail address: benedetto.intrigila@uniroma2.it

[b] Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
e-mail address: gmanzone@irif.fr

[c] USPN, LIPN, UMR 7030, CNRS, F-93430 Villetaneuse, France
e-mail address: manzonetto@univ-paris13.fr, munnich@lipn.univ-paris13.fr

Abstract. Extended addressing machines (EAMs) have been introduced to represent higher-order sequential computations. Previously, we have shown that they are capable of simulating—via an easy encoding—the operational semantics of PCF, extended with explicit substitutions. In this paper we prove that the simulation is actually an equivalence: a PCF program terminates in a numeral exactly when the corresponding EAM terminates in the same numeral. It follows that the model of PCF obtained by quotienting typable EAMs by a suitable logical relation is adequate. From a definability result stating that every EAM in the model can be transformed into a PCF program with the same observational behavior, we conclude that the model is fully abstract for PCF.

1. Introduction

The problem of determining when two programs are equivalent is central in computer science. For instance, it is necessary to verify that the optimizations performed by a compiler actually preserve the meaning of the input program. In λ-calculi like Plotkin's PCF [Plo77], two programs P_1, P_2 are considered observationally equivalent, written $P_1 \equiv_{\text{obs}} P_2$, whenever it is possible to plug either P_1 or P_2 into any context $\mathrm{C}\Box$ of appropriate type without noticing any difference in the global behavior, i.e., $\mathrm{C}[P_1]$ of type int produces a numeral $\underline{k}$ exactly when $\mathrm{C}[P_2]$ does. The Full Abstraction Problem, raised by Milner in [Mil77], aims at finding a syntax-independent description of $\equiv_{\text{obs}}$ by means of denotational models. The quest for a fully abstract (*FA*, for short) model of PCF kept researchers busy for decades [Cur07], as the usual Scott-continuous models did not enjoy this property, and culminated with FA models based on game semantics [AMJ94, Nic94, HO00]. Subsequently, a FA model of PCF based on realizability techniques was presented in [MRS99].

Key words and phrases: Extended Addressing Machines, explicit substitutions, PCF, definability, full abstraction.

Nicolas Münnich is partly supported by ANR JCJC Project CoGITARe, ANR-18-CE25-0001.

DOI:10.46298/LMCS-21(3:17)2025

Addressing machines. In 2020, the first author initiated a research program focused on modeling higher-order sequential computations through *addressing machines* (AMs), originally introduced by Della Penna in [Del97]. The intent is not to develop a denotational model grounded in abstract mathematical structures but rather to propose a computational model, alternative to the von Neumann architecture, where computation is driven by communication between machines instead of local operations. Addressing machines owe their name to the fact that they operate exclusively by manipulating the addresses of other machines (just like pure λ-terms operate on other λ-terms rather than on ordinary data types). Higher-order computations are realized by passing functional programs via these addresses. An AM can read an address from its tape, store the result of applying an address to another and pass the execution to another machine by calling its address. A challenge in defining addressing machines (AMs) lies in the fact that an address uniquely identifies a machine in a specific *state of execution*, meaning its registers and tape may contain the addresses of other AMs. The problem is finding a way to assign addresses to machines, and to compute the address resulting from applying an address a to an address b, while avoding a circular definition. The definition of AMs resolves this circularity issue by initially defining the machines over an arbitrary set $\mathbb{A}$ of addresses, and then employing external maps—whose existence is guaranteed by set-theoretical principles—to establish these associations.

Addressing machines as a model of computation. Before delving deeper into the formalism, we would like to justify AMs as a computational model and to explain the motivation behind their inception. We begin by observing that many significant results in recursion theory are obtained by passing a Turing Machine as input to another Turing Machine (possibly itself). For example, this technique is a key component in formulating the Halting Problem and proving its undecidability [Dav58, Tur36]. Since Turing Machines accept only natural numbers as input, this phenomenon of self-referentiation is achievable only through appropriate encodings. These encodings are also necessary to represent higher order computations, but difficult to handle in practice. In fact, while efficient encodings of the λ-calculus into Turing machines are well-documented in the literature [ALV23], the reverse encoding is part of the *folklore* and considered excessively verbose to be presented in detail. By making the addresses first class citizen and by explicitly and uniformly relying on external mechanisms, the framework of AMs reduces to a minimum the technicalities concerning the encodings. This is the main feature that allowed Della Penna *et al.* to define a reasonably elegant translation from λ-terms to AMs, and to show that the associated machine faithfully represents the behavior of the original λ-term [DIM22].

A corollary of the λ-definability result in [DIM22] is that all numerical partial computable functions can be represented via AMs. As previously mentioned, the definition of addressing machines depends however on the existence of an external function, called *address table map*, bijectively associating each AM with its address. Since the set of machines and the set of addresses are both countably infinite there exist a *continuum* of address table maps, most of which cannot be effective. On the one hand, choosing a non-computable function as address table map can lead to define AMs representing non-computable functions. This situation is reminiscent of Ershov's notion of *pre-complete enumeration* [Vis80], where a function $f : A \to B$ is considered computable w.r.t. two enumerations $\nu : \mathbb{N} \to A, \mu : \mathbb{N} \to B$ of its domain and codomain, and taking non-effective enumerations can lead to represent non-computable functions. On the other hand, it is possible to construct effective maps by

employing classical Gödelization techniques, therefore the formalism of addressing machines actually deserve to be considered a model of computation.

Extended addressing machines. In [DIM22], Della Penna *et al.* successfully built a model of untyped λ-calculus based on AMs, but also realized that performing calculations on natural numbers in this formalism is as awkward as using Church numerals. To avoid this problem, we recently introduced *extended addressing machines* (EAMs) [IMM23] having additional instructions for performing basic arithmetic operations on addresses of 'numeral' machines, a recursor machine representing a fixed point combinator, and a typing algorithm. A recursor can even be programmed in the original formalism, but we avoid any dependency on *self-application* by manipulating the addressing mechanism to grant it access to *its own address*. This can be seen as a very basic version of the reflection principle of programming languages. In the present paper we construct a fully abstract model of PCF based on EAMs (Theorem 5.20). We start by recalling the definition of EAMs (Section 3), their operational semantics and type system. Then, we define a translation (Definition 4.4) constructing an EAM from every typable PCF term and prove that such a translation is type-preserving (Theorem 4.6), and the resulting EAM correctly represents the behavior of the original term:

> A PCF program of type int reduces to a numeral $\underline{k}$ exactly when the corresponding EAM reduces to the machine k representing $k \in \mathbb{N}$ (Theorem 4.10).

This equivalence is achieved using as intermediate language EPCF, i.e. a PCF extended with explicit substitutions, since their operational semantics coincide on closed programs of type int. The result can be seen as a strengthening of the main theorem in [IMM23], where only one direction was proven. As a consequence, one gets that the model constructed by equating all typable EAMs having the same observational behavior is adequate for PCF (Theorem 5.10). We achieve completeness—the other side of full abstraction—by defining a 'reverse' translation that associates a PCF program with every typable EAM, and proving that the two translations are inverses up to observational equivalence on EAMs (Theorem 5.17). We emphasize that the reverse translation is made possible by the presence of the type system: an EAM is well-typed if there exists a finite derivation that recursively associates a type with each of its components. This phenomenon suppresses all non-definable elements.

The notion of model and full abstraction. The model of PCF that we construct in this paper is based on AMs and therefore does not fit into the notion of denotational model considered in the literature, which is usually based on category theory and domain theory [Cur07]. Our model is an instance of Milner's set-theoretical definition [Mil77] and provides a syntax-independent description of PCF terms. Our full abstraction result is achieved through a quotient that somewhat mimics the observational equivalence. This approach is reminiscent of the construction of fully abstract models in categories of games, which also rely on a quotient. However, while game models use the quotient solely to achieve completeness, we leverage the same quotient to establish both soundness and completeness. The interest of this kind of results mostly lies in the *definability property* that does not hold automatically in the quotiented model.[1] We believe that our approach, based on a notion of deterministic machine, brings an original, computationally-oriented, perspective in PCF semantics: e.g., in our setting no limit construction is required to handle the fixed point

[1]Even in our case, in the untyped world, there are EAMs that are not definable by any PCF program. The machines M_n introduced in Examples 3.8(2) are witnesses of this situation.

combinator since an EAM accessing its own address is easily definable from a mathematical perspective, and can be seen as an abstract view of the usual implementation of recursion.

Related and future works. A preliminary version of addressing machines was introduced in [Del97] to model computation as communication between distinguished processes by means of their addresses. They were subsequently refined in [DIM22] with the theoretical purpose of constructing a model of λ-calculus.

Compared to other machine-based formalisms, addressing machines occupy a unique space with a fair amount of individually overlapping features, but distinct overall. Many abstract machines feature subroutines which can be called and returned from. Traditional abstract machines (KAM [Kri07], SECD [Lan64], TIM [FW87], Lazy KAM [Cre91, Lan07]), ZINC [Ler90] etc.) employ a stack to temporarily store the current state of the program during the execution of a subroutine. AMs adopt a different approach, utilizing separate AMs for each subroutine and an external address table map to allow communication among them. Evaluating a subroutine is done simply by executing the corresponding AM. Thus, to perform meaningful computations, it is necessary to consider the whole set of AMs, rather than a single machine. By delegating the complexity of subroutines to the external map, AMs are difficult to compare with other formalisms in terms of implementational complexity and efficiency. Due to space limitations, any implementation can represent only a finite number of AMs and relies on a finite address table map, which is therefore decidable. From a theoretical perspective, we believe that a translation from AMs into (multiple) traditional abstract machines could be defined without much difficulty. Additionally, we emphasize that computational efficiency can be enhanced by optimizing the finite address table map through the use of suitable hash tables. Addressing this optimization challenge is a goal we plan to explore in future work.

The presence of explicit substitutions may also remind of more "modern" abstract machines arising from the linear substitution calculus, namely the MAM [ABM14, AB17] and related machines. The MAM also avoids the concept of a closure, through use of α-renaming all environments which would be local to a subroutine can instead be stored in a single global environment. One could argue that this is a special method of implementing subroutines, while AMs retain a local environment and sidestep the issue entirely.

The communication aspect of addressing machines may also remind the reader of formalisms such as Interactive Turing Machines [GMR89, Can13] and the π-calculus [Mil99, SW01] which both feature communication between processes. Both of these formalisms use the communication aspect for the purpose of parallel/concurrent computations. By design, AMs do not support concurrent computations and instead model asymmetric communication rather than bilateral communication, so the similarity ends there.

Compared with models of PCF based on Scott-continuous functions [Mil77, BCL85, Cur07], EAMs provide a more operational interpretation of a program and naturally avoid parallel features that would lead to the failure of FA as in the continuous semantics. Compared with Curien's sequential algorithms [Cur92] and categories of games [AMJ94, Nic94, HO00] they share the intensionality of the denotations of the programs, while presenting an original way of modelling sequential computation. The model based on AMs also bares *some* similarities with the categories of assembly used to model PCF [Lon95], mostly on a philosophical level, in the sense that these models are based on the 'codes' (rather than addresses) of recursive functions realizing a formula ($\cong$ type).

A disclaimer concerning explicit substitutions [ACCL90, ACCL91, CHL96, SI96, LM99]: since the full abstraction property is defined on *closed* PCF terms, we assume a closedness condition on explicit substitutions, as in [AFFM07], to reduce the level of technicalities. We do not claim, or believe, that our presentation of EPCF gives a satisfying theory of explicit substitution for 'open' PCF. In future works, we plan to build from Accattoli and Kesner's works [AK12, Acc18] to develop a version of EPCF at the correct level of generality.

2. PCF with Explicit Substitutions

We begin by describing the syntax and operational semantics of Plotkin's PCF [Plo77, Ong95].

Definition 2.1. (1) Let us consider fixed a countably infinite set Var of variables that are denoted $x, y, z, \ldots$ The set Λ^{PCF} of PCF *terms* is defined by induction as follows:

$$P, Q, Q' \ ::= x \mid \lambda x.P \mid P \cdot Q \mid \mathbf{fix}\, P \mid \mathbf{0} \mid \mathbf{pred}\, P \mid \mathbf{succ}\, P \mid \mathbf{ifz}(P, Q, Q') \qquad (\Lambda^{\mathsf{PCF}})$$

where $\lambda x.P$ represents the abstraction, $P \cdot Q$ the application, **fix** a fixed point combinator, **0** the constant zero, **pred** and **succ** the predecessor and successor (respectively), and **ifz** the conditional test on zero (namely, 'is zero?-then-else').

(2) A PCF *value* is either a numeral or an abstraction. The *set of* PCF *values* is defined by:

$$\mathrm{Val}^{\mathsf{PCF}} = \{\underline{n} \mid n \in \mathbb{N}\} \cup \{\lambda x.P \mid P \in \Lambda^{\mathsf{PCF}}\}, \text{ where } \underline{n} = \mathbf{succ}^n(\mathbf{0}).$$

(3) The set $\mathrm{FV}(P)$ of *free variables of* $P \in \Lambda^{\mathsf{PCF}}$ and *α-conversion* are defined as usual.

(4) A term $P \in \Lambda^{\mathsf{PCF}}$ is called a PCF *program* if it is closed, i.e. $\mathrm{FV}(P) = \emptyset$.

(5) Given $P, Q \in \Lambda^{\mathsf{PCF}}$, we denote by $P[Q/x]$ the *capture-free substitution* of Q for all free occurrences of x in P.

Hereafter, PCF terms are considered up to α-conversion. We now introduce PCF 'contexts', namely PCF terms possibly containing occurrences of an algebraic variable, which is traditionally called *hole*, and here denoted by $\Box$.

Definition 2.2. (1) PCF *contexts* $\mathrm{C}\Box$ are generated by the following grammar:

$$\mathrm{C}\Box \ ::= \Box \mid x \mid \lambda x.\mathrm{C} \mid \mathrm{C}_1 \cdot \mathrm{C}_2 \mid \mathbf{fix}\, \mathrm{C} \mid \mathbf{0} \mid \mathbf{pred}\, \mathrm{C} \mid \mathbf{succ}\, \mathrm{C} \mid \mathbf{ifz}(\mathrm{C}, \mathrm{C}_1, C_2)$$

(2) Given a PCF context $\mathrm{C}\Box$ and $P \in \Lambda^{\mathsf{PCF}}$, $\mathrm{C}[P]$ stands for the PCF term obtained by substituting P for all occurrences of $\Box$ in C, possibly with capture of free variables in P.

(3) PCF *evaluation contexts* are PCF contexts generated by the grammar (for $P, Q \in \Lambda^{\mathsf{PCF}}$):

$$\mathrm{E}\Box ::= \Box \mid \mathrm{E} \cdot P \mid \mathbf{pred}\, \mathrm{E} \mid \mathbf{succ}\, \mathrm{E} \mid \mathbf{ifz}(\mathrm{E}, P, Q)$$

We introduce the 'small step' and 'big step' call-by-name operational semantics of PCF.

Definition 2.3. (1) The *weak head reduction* $\to_{\mathsf{PCF}} \subseteq \Lambda^{\mathsf{PCF}} \times \Lambda^{\mathsf{PCF}}$ is defined as follows:

$$\begin{array}{ll} \mathrm{E}[(\lambda x.P)Q] \to_{\mathsf{PCF}} \mathrm{E}[P[Q/x]], & \mathrm{E}[\mathbf{fix}\,(P)] \to_{\mathsf{PCF}} \mathrm{E}[P \cdot (\mathbf{fix}\,(P))], \\ \mathrm{E}[\mathbf{pred}\,(\mathbf{succ}\,(\underline{n}))] \to_{\mathsf{PCF}} \mathrm{E}[\underline{n}], & \mathrm{E}[\mathbf{pred}\,(\mathbf{0})] \to_{\mathsf{PCF}} \mathrm{E}[\mathbf{0}], \\ \mathrm{E}[\mathbf{ifz}(\mathbf{0}, P_1, P_2)] \to_{\mathsf{PCF}} \mathrm{E}[P_1], & \mathrm{E}[\mathbf{ifz}(\underline{n+1}, P_1, P_2)] \to_{\mathsf{PCF}} \mathrm{E}[P_2], \end{array}$$

for all evaluation contexts $\mathsf{E}\Box$.

(2) The *multistep reduction* $\twoheadrightarrow_{\mathsf{PCF}}$ is defined as the reflexive and transitive closure of $\to_{\mathsf{PCF}}$.

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathsf{int}}\ (0) \qquad \frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash P : \alpha \to \alpha}{\Gamma \vdash \mathbf{fix}\, P : \alpha}\ (\mathrm{Y})$$

$$\frac{\Gamma \vdash P : \mathsf{int}}{\Gamma \vdash \mathbf{pred}\, P : \mathsf{int}}\ (-) \qquad \frac{\Gamma \vdash P : \mathsf{int}}{\Gamma \vdash \mathbf{succ}\, P : \mathsf{int}}\ (+) \qquad \frac{\Gamma, x : \alpha \vdash P : \beta}{\Gamma \vdash \lambda x.P : \alpha \to \beta}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash P : \alpha \to \beta \quad \Gamma \vdash Q : \alpha}{\Gamma \vdash P \cdot Q : \beta}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma \vdash P : \mathsf{int} \quad \Gamma \vdash Q : \alpha \quad \Gamma \vdash Q' : \alpha}{\Gamma \vdash \mathbf{ifz}(P, Q, Q') : \alpha}\ (\mathrm{ifz})$$

Figure 1: PCF type assignment system.

(3) The *big step reduction* $\Downarrow \subseteq \Lambda^{\mathsf{PCF}} \times \Lambda^{\mathsf{PCF}}$ takes a PCF term to a PCF value:

$$\frac{U \in \mathrm{Val}}{U \Downarrow U}\ (\mathrm{val}) \qquad \frac{P \Downarrow \mathbf{0}}{\mathbf{pred}\, P \Downarrow \mathbf{0}}\ (\mathrm{pr}_0) \qquad \frac{P \Downarrow \underline{n+1}}{\mathbf{pred}\, P \Downarrow \underline{n}}\ (\mathrm{pr})$$

$$\frac{P \Downarrow \mathbf{0} \quad Q \Downarrow U_1}{\mathbf{ifz}(P, Q, Q') \Downarrow U_1}\ (\mathrm{ifz}_0) \qquad \frac{P \Downarrow \underline{n+1} \quad Q' \Downarrow U_2}{\mathbf{ifz}(P, Q, Q') \Downarrow U_2}\ (\mathrm{ifz}_{>0}) \qquad \frac{P \Downarrow \underline{n}}{\mathbf{succ}\, P \Downarrow \underline{n+1}}\ (\mathrm{sc})$$

$$\frac{P \cdot (\mathbf{fix}\, P) \Downarrow U}{\mathbf{fix}\, P \Downarrow U}\ (\mathrm{fix}) \qquad \frac{P \Downarrow \lambda x.P' \quad P'[Q/x] \Downarrow U}{P \cdot Q \Downarrow U}\ (\beta_v)$$

It is well-known that the big-step and small-step semantics of PCF are equivalent on PCF programs. For a proof of this equivalence, see e.g. [Ong95].

Examples 2.4. The following PCF programs will be used as running examples:
(1) $\mathbf{I} = \lambda x.x$, representing the identity.
(2) $\mathbf{\Omega} = \mathbf{fix}\,(\mathbf{I})$, representing the paradigmatic looping program.
(3) $\mathbf{succ1} = \lambda x.\mathbf{succ}\, x$, representing the successor function.
(4) $\mathbf{succ2} = (\lambda sn.s \cdot (s \cdot n)) \cdot \mathbf{succ1}$, representing the function $f(x) = x + 2$.

Definition 2.5. (1) The set $\mathbb{T}$ of *(simple) types* over a *ground type* int is defined by:

$$\alpha, \beta \ ::= \ \mathsf{int} \mid \alpha \to \beta \qquad (\mathbb{T})$$

The arrow operator associates to the right, in other words we write $\alpha_1 \to \cdots \to \alpha_n \to \beta$ for $\alpha_1 \to (\cdots(\alpha_n \to \beta)\cdots)$ $(= \vec{\alpha} \to \beta$, for short). If $n = 0$ then $\alpha_1 \to \cdots \to \alpha_n \to \beta = \beta$.
(2) A *typing environment* is defined as a finite map $\Gamma : \mathrm{Var} \to \mathbb{T}$.
(3) We write $x_1 : \alpha_1, \ldots, x_n : \alpha_n$ for the unique environment Γ satisfying $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$ and $\Gamma(x_i) = \alpha_i$, for all $i \in \{1, \ldots, n\}$.
(4) *Typing judgements* are triples, denoted $\Gamma \vdash P : \alpha$, where Γ is a typing environment, P is a PCF term and $\alpha \in \mathbb{T}$.
(5) *Typing derivations* are finite trees built bottom-up in such a way that the root has shape $\Gamma \vdash P : \alpha$ and every node is an instance of a rule from Figure 1. In the rule $(\to_{\mathrm{I}})$ we assume wlog that $x \notin \mathrm{dom}(\Gamma)$, by α-conversion.
(6) When writing $\Gamma \vdash P : \alpha$, we mean that this typing judgement is derivable.
(7) We say that P *is typable* if $\Gamma \vdash P : \alpha$ is derivable for some Γ, α.

Examples 2.6. The following are examples of derivable typing judgments.
(1) $\vdash \lambda x.x : \alpha \to \alpha$, for all $\alpha \in \mathbb{T}$.
(2) $\vdash (\lambda sn.s \cdot (s \cdot n)) \cdot (\lambda x.\mathbf{succ}\, x) : \mathsf{int} \to \mathsf{int}$.
(3) $\vdash \mathbf{fix}\,(\lambda fxy.\mathbf{ifz}(y, x, f \cdot (\mathbf{succ}\, x) \cdot (\mathbf{pred}\, y))) : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$.
(4) $\vdash \mathbf{\Omega} : \alpha$, for all $\alpha \in \mathbb{T}$. In particular, we have $\vdash \mathbf{\Omega} : \mathsf{int}$.

2.1. PCF **with explicit substitutions.** We introduce the language EPCF, a version of PCF endowed with (closed) explicit substitutions.

Definition 2.7. (1) The set Λ^{E} of EPCF *terms* is defined by the grammar (for $x \in \mathrm{Var}$):

$$\begin{array}{rcl} L, M, N & ::= & x \mid M \cdot N \mid \lambda x.M \mid \mathbf{fix}\, M \mid M\langle N/x\rangle \mid \\ & & \mid \mathbf{0} \mid \mathbf{pred}\, M \mid \mathbf{succ}\, M \mid \mathbf{ifz}(L, M, N) \end{array} \qquad (\Lambda^{\mathsf{E}})$$

The only change compared to standard PCF is the addition of a constructor $M\langle N/x\rangle$ representing the term M receiving the *explicit substitution* $\langle N/x\rangle$ of x by N. Notice that, while $M[N/x]$ is a meta-notation, explicit substitutions are actual constructors.

(2) When considering a (possibly empty) list of explicit substitutions $\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$, we assume that the variables $\vec{x}$ in σ are pairwise distinguished. Given σ as above, we let $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$ be the *domain of* σ and write $\sigma(x_i) = N_i$, for $x_i \in \mathrm{dom}(\sigma)$.

(3) Given $M \in \Lambda^{\mathsf{E}}$, we write M^σ for $M\langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle = (\cdots(M\langle N_1/x_1\rangle)\cdots)\langle N_n/x_n\rangle$.

(4) The set $\mathrm{Val}^{\mathsf{E}}$ of EPCF *values* contains abstractions under substitutions and numerals, i.e.

$$\mathrm{Val}^{\mathsf{E}} = \{\underline{n} \mid n \in \mathbb{N}\} \cup \{(\lambda x.M)^\sigma \mid M \in \Lambda^{\mathsf{E}}\}$$

(5) Given $M \in \Lambda^{\mathsf{E}}$, the set $\mathrm{FV}(M)$ of all *free variables of* M and α*-conversion* are defined by induction on M as expected, namely with the following additional cases (for y fresh):

$$\mathrm{FV}(M\langle N/x\rangle) = (\mathrm{FV}(M) - \{x\}) \cup \mathrm{FV}(N), \qquad M\langle N/x\rangle =_\alpha M[y/x]\langle N/y\rangle,$$

where y is a fresh variable and $M[L/x]$ denotes the *capture-free substitution* of a term L (in the case above, one takes $L = y$) for all free occurrences of x in M.

(6) An EPCF term M is called *an* EPCF *program*, written $M \in \mathcal{P}_{\mathsf{E}}$, if it is closed and all its explicit substitutions are of shape $\langle N/x\rangle$ for some $x \in \mathrm{Var}$ and N closed.

(7) EPCF *evaluation contexts* are defined identically to PCF evaluation contexts (Def. 2.2(1)), but with P, Q ranging over Λ^{E}. Similarly, given $M \in \Lambda^{\mathsf{E}}$ and an evaluation context $\mathrm{E}\Box$, we write $\mathrm{E}[M]$ for the EPCF term obtained by substituting M for the hole $\Box$ in $\mathrm{E}\Box$.

Notice that, in an EPCF program, all subterms of the form $M\langle N/x\rangle$ must have $N \in \mathcal{P}_{\mathsf{E}}$. Clearly $\Lambda^{\mathsf{PCF}} \subsetneq \Lambda^{\mathsf{E}}$, moreover all PCF programs belong to $\mathcal{P}_{\mathsf{E}}$. In this paper we focus on (E)PCF programs and adopt Barendregt's *variable convention* stating that bound variables are given maximally distinguished names.

Examples 2.8. All PCF terms introduced previously are also EPCF terms. Some examples of EPCF programs that are not PCF terms are $x\langle\underline{1}/x\rangle$ and $\lambda x.(\mathbf{ifz}(x, y, z)\langle\underline{1}/y\rangle\langle\underline{2}/z\rangle)$.

We endow EPCF with a small-step call-by-name operational semantics capturing weak head reduction. The reduction is 'weak' since it does not reduce under abstractions.

Definition 2.9. (1) The *computation reduction* $\to_{\mathsf{cr}}$ on EPCF terms is defined as:

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{E}[(\lambda x.M)^\sigma N] & \to_{\mathsf{cr}} & \mathrm{E}[M^\sigma\langle N/x\rangle], & \mathrm{E}[\mathbf{pred}\,\mathbf{0}] & \to_{\mathsf{cr}} & \mathrm{E}[\mathbf{0}], \\ \mathrm{E}[\mathbf{ifz}(\mathbf{0}, M, N)] & \to_{\mathsf{cr}} & \mathrm{E}[M], & \mathrm{E}[\mathbf{pred}\,(\underline{n+1})] & \to_{\mathsf{cr}} & \mathrm{E}[\underline{n}], \\ \mathrm{E}[\mathbf{ifz}(\underline{n+1}, M, N)] & \to_{\mathsf{cr}} & \mathrm{E}[N], & \mathrm{E}[\mathbf{fix}\,(M)] & \to_{\mathsf{cr}} & \mathrm{E}[M(\mathbf{fix}\,(M))]. \end{array}$$

(2) The *percolation reduction* $\to_{\mathsf{pr}}$ on EPCF terms is defined as (where σ is non-empty):

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{E}[x^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[N], \text{ if } \sigma(x) = N, & \mathrm{E}[\mathbf{0}^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[\mathbf{0}], \\ \mathrm{E}[y^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[y], \text{ if } y \notin \mathrm{dom}(\sigma), & \mathrm{E}[(M \cdot N)^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[M^\sigma \cdot N^\sigma], \\ \mathrm{E}[(\mathbf{pred}\,(M))^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[\mathbf{pred}\,(M^\sigma)], & \mathrm{E}[(\mathbf{succ}\,(M))^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[\mathbf{succ}\,(M^\sigma)], \\ \mathrm{E}[(\mathbf{ifz}(L, M, N))^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[\mathbf{ifz}(L^\sigma, M^\sigma, N^\sigma)], & \mathrm{E}[(\mathbf{fix}\,(M))^\sigma] & \to_{\mathsf{pr}} & \mathrm{E}[\mathbf{fix}\,(M^\sigma)]. \end{array}$$

(3) The (one step) *weak head (w.h.) reduction* $\to_{\mathtt{wh}}$ is defined as the union of $\to_{\mathsf{cr}}$ and $\to_{\mathsf{pr}}$.
(4) As it is customary, we denote by $\twoheadrightarrow_{\mathtt{wh}}$ the transitive-reflexive closure of $\to_{\mathtt{wh}}$.
(5) We define $\leftrightarrow_{\mathtt{wh}}$ as the symmetric, transitive and reflexive closure of $\to_{\mathtt{wh}}$.

Remarks 2.10. (1) The reduction $\to_{\mathtt{wh}}$ is designed to work on EPCF programs. On closed terms that are not programs, we might obtain unsound reductions like the following:

$$(\lambda x.(y\langle x/y\rangle))\mathbf{0} \to_{\mathtt{wh}} y\langle x/y\rangle\langle \mathbf{0}/x\rangle \to_{\mathtt{wh}} x$$

(2) The set $\mathcal{P}_{\mathsf{E}}$ is closed under $\to_{\mathtt{wh}}$, hence under $\twoheadrightarrow_{\mathtt{wh}}$ as well.
(3) The w.h. reduction is deterministic: $N_1 \,{}_{\mathtt{wh}}\!\leftarrow M \to_{\mathtt{wh}} N_2$ implies $N_1 = N_2$.
(4) An EPCF program M is w.h.-normalizing if and only if $M \twoheadrightarrow_{\mathtt{wh}} V$, for some $V \in \mathrm{Val}^{\mathsf{E}}$.

Since $\to_{\mathtt{wh}}$ is deterministic, we can safely write $|M \twoheadrightarrow_{\mathtt{wh}} N|$ to denote the length n of the unique reduction sequence $M = M_1 \to_{\mathtt{wh}} M_2 \to_{\mathtt{wh}} \cdots \to_{\mathtt{wh}} M_n = N$.

Definition 2.11. From $\to_{\mathtt{wh}}$ and mirroring the big-step reduction of PCF, we can derive a big-step reduction $\Downarrow^{\mathsf{E}} \subseteq \Lambda^{\mathsf{EPCF}} \times \mathrm{Val}^{\mathsf{E}}$ relating an EPCF term with an EPCF value:

$$\frac{\underline{n} \in \mathbb{N}}{(\underline{n})^\sigma \Downarrow^{\mathsf{E}} \underline{n}}\ (\mathrm{nat}^{\mathsf{E}}) \qquad \frac{}{(\lambda x.M)^\sigma \Downarrow^{\mathsf{E}} (\lambda x.M)^\sigma}\ (\lambda^{\mathsf{E}}) \qquad \frac{\sigma(x) = N \quad N \Downarrow^{\mathsf{E}} V}{x^\sigma \Downarrow^{\mathsf{E}} V}\ (\mathrm{var}^{\mathsf{E}})$$

$$\frac{M^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}}{(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}}\ (\mathrm{pr}_0{}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} \underline{n+1}}{(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n}}\ (\mathrm{pr}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} \underline{n}}{(\mathbf{succ}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n+1}}\ (\mathrm{sc}^{\mathsf{E}})$$

$$\frac{L^\sigma \Downarrow^{\mathsf{E}} \mathbf{0} \quad M^\sigma \Downarrow^{\mathsf{E}} V_1}{(\mathbf{ifz}(L,M,N))^\sigma \Downarrow^{\mathsf{E}} V_1}\ (\mathrm{ifz}_0{}^{\mathsf{E}}) \qquad \frac{L^\sigma \Downarrow^{\mathsf{E}} \underline{n+1} \quad N^\sigma \Downarrow^{\mathsf{E}} V_2}{(\mathbf{ifz}(L,M,N))^\sigma \Downarrow^{\mathsf{E}} V_2}\ (\mathrm{ifz}_{>0}{}^{\mathsf{E}})$$

$$\frac{M^\sigma \cdot \mathbf{fix}\,(M^\sigma) \Downarrow^{\mathsf{E}} V}{(\mathbf{fix}\, M)^\sigma \Downarrow^{\mathsf{E}} V}\ (\mathrm{fix}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} (\lambda x.M')^{\sigma'} \quad (M')^{\sigma'}\langle N^\sigma/x\rangle \Downarrow^{\mathsf{E}} V}{(M \cdot N)^\sigma \Downarrow^{\mathsf{E}} V}\ (\beta_v^{\mathsf{E}})$$

Proposition 2.12. *Given an* EPCF *program* M *and an* EPCF *value* V*, we have*

$$M \Downarrow^{\mathsf{E}} V \iff M \twoheadrightarrow_{\mathtt{wh}} V.$$

Proof (Appendix A.1). ($\Rightarrow$) By induction on the height of a derivation of $M \Downarrow^{\mathsf{E}} V$.
($\Leftarrow$) By induction on the length $|M \twoheadrightarrow_{\mathtt{wh}} V|$. □

EPCF terms can be typed in a manner very similar to PCF terms.

Definition 2.13. (1) An EPCF typing judgement is, like in PCF, a triple of $\Gamma \vdash M : \alpha$.
(2) The rules for typing an EPCF term have been presented in Figure 2.

The only change compared to PCF typing rules is the introduction of the (σ) rule. In this rule, we rely on the fact that in a program of shape $M\langle N/x\rangle$, the subterm N must be closed and remove the typing environment in $\vdash N : \beta$. This is not standard when considering more general notions of explicit substitutions, but simplifies our definitions later.

Remark 2.14. This approach of forcing explicit substitutions to be closed also sidesteps the issues caused by Melliès' degeneracy [Mel95].

Example 2.15. The running examples can be typed without the use of the (σ) rule, as they are PCF terms. For an EPCF exclusive term, as an example, we type $x\langle\mathbf{succ}\,\mathbf{0}/x\rangle$:

$$\dfrac{\dfrac{}{x : \mathsf{int} \vdash x : \mathsf{int}}\ (\mathrm{ax}) \quad \dfrac{\dfrac{}{\vdash \mathbf{0} : \mathsf{int}}\ (0)}{\vdash \mathbf{succ}\,\mathbf{0} : \mathsf{int}}\ (+)}{\vdash x\langle\mathbf{succ}\,\mathbf{0}/x\rangle : \mathsf{int}}\ (\sigma)$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathsf{int}}\ (0) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash M \cdot N : \beta}\ (\to_{\mathrm{E}}) \qquad \frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\mathrm{ax})$$

$$\frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{succ}\, M : \mathsf{int}}\ (+) \qquad \frac{\Gamma, x : \beta \vdash M : \alpha \quad \vdash N : \beta}{\Gamma \vdash M\langle N/x\rangle : \alpha}\ (\sigma) \qquad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{pred}\, M : \mathsf{int}}\ (-) \qquad \frac{\Gamma \vdash L : \mathsf{int} \quad \Gamma \vdash M : \alpha \quad \Gamma \vdash N : \alpha}{\Gamma \vdash \mathbf{ifz}(L, M, N) : \alpha}\ (\mathrm{ifz}) \qquad \frac{\Gamma \vdash M : \alpha \to \alpha}{\Gamma \vdash \mathbf{fix}\, M : \alpha}\ (\mathrm{Y})$$

Figure 2: EPCF type assignment system. In the rule (σ) we enforce N to be closed.

The following lemma summarizes the main properties of the type assignment system.

Lemma 2.16. *Let $M \in \Lambda^{\mathsf{E}}$, $\alpha, \beta \in \mathbb{T}$ and Γ be a typing environment.*

(1) *(Syntax directedness) Every derivable judgement $\Gamma \vdash M : \alpha$ admits a unique derivation.*
(2) *(Strengthening) $\Gamma, x : \beta \vdash M : \alpha$ and $x \notin \mathrm{FV}(M)$ hold if and only if $\Gamma \vdash M : \alpha$ does. Thus, an EPCF program M is typable if it is typable in the empty environment.*
(3) *(Subject reduction) For all $M \in \mathcal{P}_{\mathsf{E}}$, $\vdash M : \alpha$ and $M \to_{\mathtt{wh}} N$ entail $\vdash N : \alpha$.*

We now move on to proving that PCF and EPCF operational semantics coincide on closed terms of type int. To this purpose, we first introduce the *collapse* $M^\dagger$ of an EPCF term M defined by performing all of the internal explicit substitutions.

Definition 2.17. (1) Given an EPCF term M, define a PCF term $M^\dagger \in \Lambda^{\mathsf{PCF}}$ as follows:

$$\begin{array}{rclcrcl} x^\dagger & = & x & & \mathbf{0}^\dagger & = & \mathbf{0} \\ (M \cdot N)^\dagger & = & M^\dagger \cdot N^\dagger & & (\mathbf{pred}\, M)^\dagger & = & \mathbf{pred}\, M^\dagger \\ (\lambda x.M)^\dagger & = & \lambda x.M^\dagger & & (\mathbf{succ}\, M)^\dagger & = & \mathbf{succ}\, M^\dagger \\ (\mathbf{fix}\, M)^\dagger & = & \mathbf{fix}\, (M^\dagger) & & (\mathbf{ifz}(L, M, N))^\dagger & = & \mathbf{ifz}(L^\dagger, M^\dagger, N^\dagger) \\ (M\langle N/x\rangle)^\dagger & = & M^\dagger[N^\dagger/x] & & & & \end{array}$$

(2) The *head size* $\lfloor - \rfloor : \Lambda^{\mathsf{E}} \to \mathbb{N}$ of an EPCF term is defined as follows:

$$\begin{array}{rclcrcl} \lfloor x \rfloor = \lfloor \mathbf{0} \rfloor & = & 1 & & \lfloor M\langle N/x\rangle \rfloor & = & \lfloor M \rfloor \cdot (\lfloor N \rfloor + 1) \\ \lfloor M \cdot N \rfloor & = & \lfloor M \rfloor + 1 & & \lfloor \lambda x.M \rfloor & = & \lfloor M \rfloor + 1 \\ \lfloor \mathbf{pred}\, M \rfloor & = & \lfloor M \rfloor + 1 & & \lfloor \mathbf{fix}\, M \rfloor & = & \lfloor M \rfloor + 1 \\ \lfloor \mathbf{succ}\, M \rfloor & = & \lfloor M \rfloor + 1 & & \lfloor \mathbf{ifz}(L, M, N) \rfloor & = & \lfloor L \rfloor + 1 \end{array}$$

(3) The map $\lfloor - \rfloor$ is extended to explicit substitutions $\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$, by setting

$$\lfloor \sigma \rfloor = \prod_{i=1}^{n} (\lfloor N_i \rfloor + 1).$$

Recall that $\mathcal{P}_{\mathsf{E}}$ denotes the set of EPCF programs, i.e. the set of closed EPCF terms M whose explicit substitutions (if any) are of the form $\langle N/x\rangle$ for N closed.

Lemma 2.18. (1) *If $M \in \mathcal{P}_{\mathsf{E}}$ then $M^\dagger$ is a PCF program.*
(2) *If $P \in \Lambda^{\mathsf{PCF}}$ then $P^\dagger = P$.*

Proof. (1) By structural induction on M.
(2) By structural induction on P. □

For a proof of the following proposition, we refer to the technical Appendix A.1.

Proposition 2.19. (1) *Let $M, N \in \mathcal{P}_{\mathsf{E}}$ be such that $M \to_{\mathsf{cr}} N$. Then $M^\dagger \to_{\mathsf{PCF}} N^\dagger$.*
(2) *Let $M, N \in \mathcal{P}_{\mathsf{E}}$ be such that $M \to_{\mathsf{pr}} N$. Then $M^\dagger = N^\dagger$.*

Corollary 2.20. *Let $M \in \mathcal{P}_{\mathsf{E}}$ be such that $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$. Then $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$.*

Proof. By induction on the length of the reduction sequence $\ell = |M \twoheadrightarrow_{\mathtt{wh}} \underline{n}|$.

Case $\ell = 0$. Then $M = \underline{n} = M^\dagger$, so this case follows by reflexivity of $\twoheadrightarrow_{\mathsf{PCF}}$.

Case $\ell > 0$. Then there exists $N \in \mathcal{P}_{\mathsf{E}}$ such that $M \to_{\mathtt{wh}} N \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ where $|N \twoheadrightarrow_{\mathtt{wh}} \underline{n}| < \ell$. By Proposition 2.19, we have $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} N^\dagger$. By induction hypothesis, we obtain $N^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. By transitivity, we conclude $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. □

Lemma 2.21. *The percolation reduction $\to_{\mathsf{pr}}$ on EPCF terms is strongly normalizing. More precisely:*

$$M \to_{\mathsf{pr}} N \;\Rightarrow\; \lfloor M \rfloor > \lfloor N \rfloor$$

Proof. By induction on a derivation of $M \to_{\mathsf{pr}} N$. In the following we consider a non-empty list of explicit substitutions $\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$, i.e. $n > 0$. It follows that $\lfloor \sigma \rfloor > 1$.

- Base cases.
  - Case $M = x_i^\sigma$ and $N = \sigma(x_i) = N_i$. Then $\lfloor x_i^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > \lfloor N_i \rfloor = \lfloor N \rfloor$.
  - Case $M = \mathbf{0}^\sigma$ and $N = \mathbf{0}$. Then $\lfloor \mathbf{0}^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > 1 = \lfloor \mathbf{0} \rfloor$.
  - Case $M = y^\sigma$, with $y \notin \mathrm{dom}(\sigma)$, and $N = y$. Then $\lfloor y^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > 1 = \lfloor y \rfloor$.
  - Case $M = (M_1 \cdot M_2)^\sigma$. Then $\lfloor (M_1 \cdot M_2)^\sigma \rfloor = (\lfloor M_1 \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M_1 \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor M_1^\sigma \cdot M_2^\sigma \rfloor$.
  - Case $M = (\mathbf{pred}\, M')^\sigma$. Then $\lfloor (\mathbf{pred}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{pred}\, (M')^\sigma \rfloor$.
  - Case $M = (\mathbf{succ}\, M')^\sigma$. Then $\lfloor (\mathbf{succ}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{succ}\, (M')^\sigma \rfloor$.
  - Case $M = (\mathbf{ifz}(M_1, M_2, M_3))^\sigma$. Then $\lfloor (\mathbf{ifz}(M_1, M_2, M_3))^\sigma \rfloor = (\lfloor M_1 \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M_1 \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{ifz}(M_1^\sigma, M_2^\sigma, M_3^\sigma) \rfloor$.
  - Case $M = (\mathbf{fix}\, M')^\sigma$. Then $\lfloor (\mathbf{fix}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{fix}\, (M')^\sigma \rfloor$.
- Induction case $M = M_1 \cdot M_2$ and $N = N_1 \cdot M_2$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $\lfloor M_1 \rfloor > \lfloor N_1 \rfloor$. Therefore $\lfloor M_1 \rfloor + 1 > \lfloor N_1 \rfloor + 1$.

The remaining cases follow analogously from the induction hypothesis. □

Proposition 2.22. *Let $(M_n)_{n \in \mathbb{N}}$ be a sequence of EPCF programs such that $M_n \to_{\mathtt{wh}} M_{n+1}$. Then for all $i \in \mathbb{N}$ there exists an index $j > i$ such that $M_i \twoheadrightarrow_{\mathtt{wh}} M_j$ and $M_i^\dagger \to_{\mathsf{PCF}} M_j^\dagger$.*

Proof. Consider an arbitrary M_i. By Lemma 2.21 $M_i \twoheadrightarrow_{\mathsf{pr}} M_k$, for some $k \geq i$ such that M_k in $\to_{\mathsf{pr}}$-normal form. Therefore $M_k \to_{\mathtt{wh}} M_{k+1}$ must be a computation step, i.e. $M_k \to_{\mathsf{cr}} M_{k+1}$. By Proposition 2.19, $M_i^\dagger = M_k^\dagger \to_{\mathsf{PCF}} M_{k+1}^\dagger$. Conclude by taking $j = k + 1 > i$. □

Theorem 2.23. *For a PCF program P having type int, we have:*

$$P \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$$

Proof. ($\Rightarrow$) We prove the contrapositive. Assume that $P \not\twoheadrightarrow_{\mathtt{wh}} \underline{n}$. By Subject Reduction (Lemma 2.16(3)) P must have an infinite $\to_{\mathtt{wh}}$ reduction path. By Proposition 2.22, $P^\dagger$ must have an infinite $\to_{\mathsf{PCF}}$ reduction path. Conclude since, by Lemma 2.18(2), $P^\dagger = P$.

($\Leftarrow$) Assume $P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$. Since P is a PCF program it also belongs to $\mathcal{P}_{\mathsf{E}}$. By Corollary 2.20 we have $P^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. Conclude since, by Lemma 2.18(2), $P^\dagger = P$. □

3. EAMs: Extended Addressing Machines

We extend the addressing machines from [DIM22] with instructions for performing arithmetic operations and conditional testing. Natural numbers are represented by particular machines playing the role of numerals.

3.1. **Main definitions.** We consider fixed a countably infinite set $\mathbb{A}$ of *addresses* together with a distinguished countable subset $\mathbb{X} \subset \mathbb{A}$, such that $\mathbb{A} - \mathbb{X}$ remains infinite. Intuitively, $\mathbb{X}$ is the set of addresses that we reserve for the numerals, therefore hereafter we work under the hypothesis that $\mathbb{X} = \mathbb{N}$, an assumption that we can make without loss of generality.

Definition 3.1. (1) Let $\varnothing \notin \mathbb{A}$ be a "null" constant representing an uninitialised register. Set $\mathbb{A}_\varnothing = \mathbb{A} \cup \{\varnothing\}$.

(2) An $\mathbb{A}$-*valued tape* T is a finite ordered list of addresses $T = [a_1, \ldots, a_n]$ with $a_i \in \mathbb{A}$ for all $i\,(1 \leq i \leq n)$. When $\mathbb{A}$ is clear from the context, we simply call T a tape. We denote by $\mathcal{T}_\mathbb{A}$ the set of all $\mathbb{A}$-valued tapes.

(3) Let $a \in \mathbb{A}$ and $T, T' \in \mathcal{T}_\mathbb{A}$. We denote by $a :: T$ the tape having a as first element and T as tail. We write $T \mathbin{@} T'$ for the concatenation of T and T', which is an $\mathbb{A}$-valued tape itself.

(4) Given a natural number $i \geq 0$, an $\mathbb{A}_\varnothing$-valued *register* R_i is a memory-cell, accessible via its index i, and capable of storing either $\varnothing$ or an address $a \in \mathbb{A}$. We write $!R_i$ to represent the value stored in the register R_i. (The notation $!R_i$ is borrowed from ML, where ! represents an explicit dereferencing operator.)

(5) Given $\mathbb{A}_\varnothing$-valued registers $R_0, \ldots, R_r$ for $r \geq 0$, an address $a \in \mathbb{A}$ and an index $i \geq 0$, we write $\vec{R}[R_i := a]$ for the list of registers $\vec{R}$ where the value of R_i has been updated by setting $!R_i = a$. Notice that, whenever $i > r$, we assume that the contents of $\vec{R}$ remains unchanged, i.e. $\vec{R}[R_i := a] = \vec{R}$.

Intuitively, the contents of the registers $R_0, \ldots, R_r$ constitute the *state* of a machine, while the tape correspond to the list of its inputs. The addressing machines from [DIM22] are endowed with only three instructions (i, j, k, l range over indices of registers):

1. `Load` i : If the tape is non-empty, 'pops' the address a from the input tape $a :: T$ and stores a in the register R_i. If the tape is empty then the machine halts its execution.
2. $k \leftarrow$ `App`(i, j) : reads the addresses a_1, a_2 from R_i and R_j respectively, and stores in R_k the address of the machine obtained by extending the tape of the machine of address a_1 with the address a_2. The resulting address is not calculated internally but rather obtained calling an external *application map*.
3. `Call` i : transfers the computation to the machine having as address the value stored in R_i, whose tape is extended with the remainder of the current machine's tape.

As a general principle, writing on a non-existing register does not cause issues as the value is simply discarded—this is in fact the way one can erase an argument. Attempts to read an uninitialized register can be avoided statically (see Lemma 3.4).

We enrich the above set of instructions with arithmetic operations mimicking the ones present in PCF:

4. $l \leftarrow$ `Test`(i, j, k): implements the "*is zero?*" test on $!R_i$. Assuming that the value of R_i is an address $n \in \mathbb{N}$, the instruction stores in R_l the value of R_j or R_k, depending on whether $n = 0$.

5. $j \leftarrow \texttt{Pred}(i)$: if $!R_i \in \mathbb{N}$, the value of R_j becomes $!R_i \ominus 1 = \max(!R_i - 1, 0)$.
6. $j \leftarrow \texttt{Succ}(i)$: if $!R_i \in \mathbb{N}$, then the value of R_j becomes $!R_i + 1$.

Notice that the instructions above need R_i to contain a natural number to perform the corresponding operation. However, they are also supposed to work on addresses of machines that compute a numeral. For this reason, the machine whose address is stored in R_i must first be executed, and only if the computation terminates with a numeral is the arithmetic operation performed. If the computation terminates in an address not representing a numeral, then the machine halts. We will see that these terminations can be avoided using a type inference algorithm (see Proposition 3.17, below).

Definition 3.2. (1) A *program* P is a finite list of instructions generated by the following grammar, where ε represents the empty string and i, j, k, l are indices of registers:

$$\begin{array}{lcl} \texttt{P} & ::= & \texttt{Load}\ i;\ \texttt{P} \mid \texttt{A} \\ \texttt{A} & ::= & k \leftarrow \texttt{App}(i, j);\ \texttt{A} \mid l \leftarrow \texttt{Test}(i, j, k);\ \texttt{A} \mid j \leftarrow \texttt{Pred}(i);\ \texttt{A} \mid j \leftarrow \texttt{Succ}(i);\ \texttt{A} \mid \texttt{C} \\ \texttt{C} & ::= & \texttt{Call}\ i \mid \varepsilon \end{array}$$

Thus, a program starts with a list of `Load`'s, continues with a list of `App`, `Test`, `Pred`, `Succ`, and possibly ends with a `Call`. Each of these lists may be empty, in particular the empty program ε can be generated.

(2) In a program, we write $\texttt{Load}\ (i_1, \ldots, i_n)$ as an abbreviation for the instruction sequence $\texttt{Load}\ i_1; \cdots ; \texttt{Load}\ i_n$. For a given instruction `ins`, we write $\texttt{ins}^a$ for the repetition of `ins` a-many times.

(3) Let P be a program, $r \geq 0$, and $\mathcal{I} \subseteq \{0, \ldots, r-1\}$ be a set of indices corresponding to the indices of initialized registers. Define the relation $\mathcal{I} \models^r P$, whose intent is to specify that P does not read uninitialized registers, as the least relation closed under the rules:

$$\frac{}{\mathcal{I} \models^r \varepsilon} \qquad \frac{i \in \mathcal{I}}{\mathcal{I} \models^r \texttt{Call}\ i} \qquad \frac{\mathcal{I} \cup \{j\} \models^r \texttt{A} \quad i \in \mathcal{I} \quad j < r}{\mathcal{I} \models^r j \leftarrow \texttt{Pred}(i);\ \texttt{A}}$$

$$\frac{\mathcal{I} \cup \{i\} \models^r \texttt{P} \quad i < r}{\mathcal{I} \models^r \texttt{Load}\ i;\ \texttt{P}} \qquad \frac{\mathcal{I} \models^r \texttt{P} \quad i \geq r}{\mathcal{I} \models^r \texttt{Load}\ i;\ \texttt{P}} \qquad \frac{\mathcal{I} \cup \{j\} \models^r \texttt{A} \quad i \in \mathcal{I} \quad j < r}{\mathcal{I} \models^r j \leftarrow \texttt{Succ}(i);\ \texttt{A}}$$

$$\frac{\mathcal{I} \cup \{k\} \models^r \texttt{A} \quad i, j \in \mathcal{I} \quad k < r}{\mathcal{I} \models^r k \leftarrow \texttt{App}(i, j);\ \texttt{A}} \qquad \frac{\mathcal{I} \cup \{l\} \models^r \texttt{A} \quad i, j, k \in \mathcal{I} \quad l < r}{\mathcal{I} \models^r l \leftarrow \texttt{Test}(i, j, k);\ \texttt{A}}$$

(4) A program P is *valid with respect to* $R_0, \ldots, R_{r-1}$ if $\mathcal{R} \models^r P$ holds for

$$\mathcal{R} = \{i \mid R_i \neq \varnothing \ \wedge\ 0 \leq i < r\}.$$

Notice that requiring all loads to occur at the beginning of the program is not an actual limitation. Once the operational semantics of AMs is discussed, we encourage the reader to verify that any AM whose program contains loads at arbitrary positions can be transformed into an equivalent AM, where all loads are placed at the beginning.

Examples 3.3. For each of these programs, we specify its validity w.r.t. $R_0 = 7$, $R_1 = a$, $R_2 = \varnothing$ (i.e., $r = 3$).

$P_1 = 2 \leftarrow \texttt{Pred}(0);\ \texttt{Call}\ 2$ (valid)
$P_2 = \texttt{Load}\ (2, 8);\ 0 \leftarrow \texttt{Test}(0, 1, 2);\ \texttt{Call}\ 0$ (valid)
$P_3 = \texttt{Load}\ (0, 2, 8);\ \texttt{Call}\ 8$ (calling the uninitialized register R_8, thus not valid)
$P_4 = 0 \leftarrow \texttt{Succ}(2);\texttt{Call}\ 1$ (reading from the uninitialized register R_2, thus not valid)
$P_5 = 8 \leftarrow \texttt{Pred}(0);\texttt{Call}\ 0$ (storing in non-existent register R_0, thus not valid)

Lemma 3.4. *Given $\mathbb{A}_\varnothing$-valued registers $\vec{R}$ and a program P it is decidable whether P is valid w.r.t. $\vec{R}$.*

Proof. Decidability follows from the syntax directedness of Definition 3.2(3), and the preservation of the invariant $\mathcal{I} \subseteq \{0, \dots, r-1\}$, since $\mathcal{I}$ is only extended with $k < r$. □

Hereafter, we will focus on EAMs having valid programs.

Definition 3.5. (1) An *extended addressing machine* (*EAM*) M over $\mathbb{A}$ (having $r+1$ registers) is given by a tuple $\mathsf{M} = \langle R_0, \dots, R_r, P, T\rangle$ where
- $\vec{R}$ are $\mathbb{A}$-valued registers,
- P is a program valid w.r.t. $\vec{R}$, and
- $T \in \mathcal{T}_\mathbb{A}$ is an (input) tape.

(2) We denote by $\mathcal{M}_\mathbb{A}$ the set of all extended addressing machines over $\mathbb{A}$.
(3) Given an EAM M, we write $\mathsf{M}.\vec{R}$ for the list of its registers, $\mathsf{M}.R_i$ for its i-th register, $\mathsf{M}.P$ for the associated program and $\mathsf{M}.T$ for its input tape.
(4) Given $\mathsf{M} \in \mathcal{M}_\mathbb{A}$ and $T' \in \mathcal{T}_\mathbb{A}$, we write $\mathsf{M} \mathbin{@} T'$ for the machine

$$\langle \mathsf{M}.\vec{R}, \mathsf{M}.P, \mathsf{M}.T \mathbin{@} T' \rangle.$$

(5) For $n \in \mathbb{N}$, the *n-th numeral machine* is defined as $\mathsf{n} = \langle R_0, \varepsilon, []\rangle$, with $!R_0 = n$.
(6) For every $a \in \mathbb{A}$, define the following extended addressing machine:

$$\mathsf{Y}^a = \langle R_0 = \varnothing, R_1 = \varnothing, \mathtt{Load}\ (0,1); 0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1, [a]\rangle.$$

We now enter into the details of the addressing mechanism which constitutes the core of this formalism.

Definition 3.6. Recall that $\mathbb{N}$ stands for an infinite subset of $\mathbb{A}$, here identified with the set of natural numbers, and that Y^a has been introduced in Definition 3.5(6).

(1) Since $\mathcal{M}_\mathbb{A}$ is countable, by a simple set-theoretical argument, we can fix a bijective function called an *address table map* (ATM)

$$\# : \mathcal{M}_\mathbb{A} \to \mathbb{A}$$

satisfying the following conditions:
  (a) (Numerals) $\forall n \in \mathbb{N} \,.\, \#\mathsf{n} = n$, where n is the n-th numeral machine;
  (b) (Fixed point combinator) $\exists a \in \mathbb{A} - \mathbb{N} \,.\, \#(\mathsf{Y}^a) = a$.

(2) We write Y for the EAM Y^a satisfying $\#(\mathsf{Y}^a) = a$ (which exists by condition (1b) above).
(3) Given $\mathsf{M} \in \mathcal{M}_\mathbb{A}$, we call the element $\#\mathsf{M}$ the *address of the EAM* M. Conversely, the EAM having as address $a \in \mathbb{A}$ is denoted by $\#^{-1}(a)$, i.e. $\#^{-1}(a) = \mathsf{M} \iff \#\mathsf{M} = a$.
(4) Define the *application map* $(\cdot) : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$ by setting

$$a \cdot b = \#(\#^{-1}(a) \mathbin{@} [b]\,).$$

I.e., the *application* of a to b is the unique address c of the EAM obtained by appending b at the end of the input tape of the EAM $\#^{-1}(a)$.

Remark 3.7. In general, there are uncountably many possible address table maps of arbitrary computational complexity. A natural example of such maps is given by *Gödelization*, which can be performed effectively. The framework is however more general and allows us to consider non-r.e. sets of addresses like the complement K^c of the halting set

$$K = \{(i, x) \mid \text{the program } i \text{ terminates when run on input } x\}$$

and a non-computable function $\# : \mathcal{M}_{K^c} \to K^c$ as a map.

In an implementation of EAMs the address table map should be computable—one can choose a fresh address from $\mathbb{A}$ whenever a new machine is constructed, save the correspondence in some table and retrieve it in constant time.

Remarks 3.8. (1) Since $\mathcal{M}_{\mathbb{A}}$ and $\mathbb{A}$ are both countable, the existence of $2^{\aleph_0}$ ATMs follows from cardinality reasons. This includes effective maps as well as non-computable ones.
(2) Depending on the chosen address table map, there might exist infinite (static) chains of EAMs, e.g., EAMs $(\mathsf{M}_n)_{n\in\mathbb{N}}$ satisfying $\mathsf{M}_n = \langle R_0, \varepsilon, [] \rangle$ with $!R_0 = \#\mathsf{M}_{n+1}$.

The results presented in this paper are independent from the choice of the ATM.

Examples 3.9. The following are examples of EAMs (whose registers are assumed uninitialized where unspecified, i.e. $\vec{R} = \vec{\varnothing}$).
(1) $\mathsf{I} := \langle R_0 = \varnothing, \texttt{Load}\ 0; \texttt{Call}\ 0, [] \rangle$,
(2) For some $a \in \mathbb{A}$, $\langle R_0 = a, R_1 = \varnothing, 0 \leftarrow \texttt{App}(1,\ 0); \texttt{Call}\ 1, [] \rangle$
(3) $\mathsf{Succ1} := \langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \rangle$.
(4) $\mathsf{Succ2} := \langle R_0, R_1, \texttt{Load}\ 0; \texttt{Load}\ 1; 1 \leftarrow \texttt{App}(0,\ 1); 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [a_{\mathsf{S}}] \rangle$, where $a_{\mathsf{S}} = \#\mathsf{Succ1}$.
(5) $\mathsf{Add\_aux} := \langle \vec{R}, P, [] \rangle$ with $\mathsf{Add\_aux}.r = 5$ and $P = \texttt{Load}\ (0, 1, 2); 3 \leftarrow \texttt{Pred}(1); 4 \leftarrow \texttt{Succ}(2); 0 \leftarrow \texttt{App}(0,\ 3);\ 0 \leftarrow \texttt{App}(0,\ 4); 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0$.

3.2. **Operational semantics.** The operational semantics of extended addressing machines is given through a small-step rewriting system. The reduction strategy is deterministic, since the only applicable rule at every step is univocally determined by the first instruction of the internal program, the contents of the registers and the head of the tape.

Definition 3.10. (1) Define a reduction strategy $\to_{\mathsf{c}}$ on EAMs, representing one step of computation, as the least relation $\to_{\mathsf{c}} \subseteq \mathcal{M}_{\mathbb{A}} \times \mathcal{M}_{\mathbb{A}}$ closed under the rules in Figure 3.
(2) The *multistep reduction* $\twoheadrightarrow_{\mathsf{c}}$ is defined as the transitive-reflexive closure of $\to_{\mathsf{c}}$.
(3) The *conversion relation* $\leftrightarrow_{\mathsf{c}}$ is the transitive-reflexive-symmetric closure of $\to_{\mathsf{c}}$.
(4) Given $\mathsf{M}, \mathsf{N}, \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$, we write $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}| \in \mathbb{N}$ for the length of the (unique) reduction path from M to N.
(5) For $n \geq 0$, we write $\mathsf{M} \twoheadrightarrow_{\mathsf{c}}^{n} \mathsf{N}$ whenever $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ and $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}| = n$ hold.

As a matter of terminology, we say that an EAM M:

- is *stuck* if its program has shape $\mathsf{M}.P = \texttt{Load}\ i; P$ but $\mathsf{M}.T = []$;
- *is in final state* if M is not in an error state, but cannot reduce further, i.e. $\mathsf{M} \not\to_{\mathsf{c}}$;
- *is in an error state* if its program has shape $\mathsf{M}.P = \texttt{ins}; P'$ for some instruction $\texttt{ins} \in \{\texttt{Load}\ i, j \leftarrow \texttt{Pred}(i), j \leftarrow \texttt{Succ}(i), l \leftarrow \texttt{Test}(i,\ j,\ k)\}$, but $!R_i \notin \mathbb{N}$ and $\#^{-1}(!R_i)$ cannot reduce further.
- *reaches a final state* (resp. *raises an error*) if $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$ for some M' in final (resp. error) state; M *does not terminate*, otherwise.

Given an EAM M, the first instruction of its program, together with the contents of its registers and tape, univocally determine which rule from Figure 3 is applicable (if any). When M tries to perform an arithmetic operation in one of its registers, say $R_i = a$, it needs to wait that the EAM $\#^{-1}(a)$ terminates its execution. If it does then the success of the operation depends on whether the result is a numeral, otherwise M is in an error state.

Unconditional rewriting rules

$$\begin{aligned}\langle \vec{R}, \texttt{Call}\ i; P, T\rangle &\to_{\mathsf{c}} \#^{-1}(!R_i) \mathbin{@} T\\ \langle \vec{R}, \texttt{Load}\ i; P, a :: T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_i := a], P, T\rangle\\ \langle \vec{R}, k \leftarrow \texttt{App}(i,\ j); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_k := !R_i \cdot !R_j], P, T\rangle\end{aligned}$$

Under the assumption that $!R_i \in \mathbb{N}$.

$$\begin{aligned}\langle \vec{R}, j \leftarrow \texttt{Pred}(i); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_j := !R_i \ominus 1, P, T\rangle, \text{ where } n \ominus 1 := \max\{n-1, 0\},\\ \langle \vec{R}, j \leftarrow \texttt{Succ}(i); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_j := !R_i + 1, P, T\rangle\\ \langle \vec{R}, l \leftarrow \texttt{Test}(i,\ j,\ k); P, T\rangle &\to_{\mathsf{c}} \begin{cases}\langle \vec{R}[R_l := !R_j], P, T\rangle, & \text{if } !R_i = 0,\\ \langle \vec{R}[R_l := !R_k], P, T\rangle, & \text{otherwise.}\end{cases}\end{aligned}$$

Under the assumption that $\#^{-1}(!R_i) \to_{\mathsf{c}} \mathsf{M}$.

$$\begin{aligned}\langle \vec{R}, j \leftarrow \texttt{Pred}(i); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_i := \#\mathsf{M}], j \leftarrow \texttt{Pred}(i); P, T\rangle\\ \langle \vec{R}, j \leftarrow \texttt{Succ}(i); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_i := \#\mathsf{M}], j \leftarrow \texttt{Succ}(i); P, T\rangle\\ \langle \vec{R}, l \leftarrow \texttt{Test}(i,\ j,\ k); P, T\rangle &\to_{\mathsf{c}} \langle \vec{R}[R_i := \#\mathsf{M}], l \leftarrow \texttt{Test}(i,\ j,\ k); P, T\rangle\end{aligned}$$

Figure 3: Small-step operational semantics for extended addressing machines.

Lemma 3.11. (1) *The strategy $\to_{\mathsf{c}}$ is deterministic:* $\mathsf{N}\ {}_{\mathsf{c}}\!\leftarrow \mathsf{M} \to_{\mathsf{c}} \mathsf{N}'$ *implies* $\mathsf{N} = \mathsf{N}'$.
(2) *The reduction $\to_{\mathsf{c}}$ is Church-Rosser:* $\mathsf{M} \leftrightarrow_{\mathsf{c}} \mathsf{N} \Leftrightarrow \exists \mathsf{Z} \in \mathcal{M}_{\mathbb{A}}\,.\,\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}\ {}_{\mathsf{c}}\!\twoheadleftarrow \mathsf{N}$.
(3) *If* $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$, *then* $\mathsf{M} \mathbin{@} [\#\mathsf{N}] \twoheadrightarrow_{\mathsf{c}} \mathsf{M}' \mathbin{@} [\#\mathsf{N}]$.

Proof. (1) Trivial: There is at most one applicable reduction rule for each state.
(2) Trivial as a consequence of (1).
(3) By induction on the length of $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$. □

Examples 3.12. See Example 3.9 for the definition of $\mathsf{I}, \mathsf{Succ1}$, $\mathsf{Succ2}$, $\mathsf{Add\_aux}$.
(1) For some $a \in \mathbb{A}$, $\mathsf{I} \mathbin{@} [a] \to_{\mathsf{c}} \langle R_0 = a, \texttt{Call}\ 0, []\rangle \to_{\mathsf{c}} \#^{-1}(a)$
(2) We have $\mathsf{Succ1} \mathbin{@} [0] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(1)$ and $\mathsf{Succ2} \mathbin{@} [1] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(3)$.
(3) Define $\mathsf{Add} = \mathsf{Y} \mathbin{@} [\#\mathsf{Add\_aux}]$, an EAM performing the addition. We show:

$$\begin{aligned}&\mathsf{Add} \mathbin{@} [1,3] \to_{\mathsf{c}} \langle (R_0 = \#\mathsf{Y}, R_1 = \#\mathsf{Add\_aux}), 0 \leftarrow \texttt{App}(0,\ 1); 1 \leftarrow \texttt{App}(1,\ 0); \texttt{Call}\ 1, [1,3]\rangle\\ &\to_{\mathsf{c}} \left\langle \vec{R}, \texttt{Load}\ (0,1,2); 3 \leftarrow \texttt{Pred}(1); 4 \leftarrow \texttt{Succ}(2); 0 \leftarrow \texttt{App}(0,\ 3); 0 \leftarrow \texttt{App}(0,\ 4);\right.\\ &\qquad\qquad \left. 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0, [\#\mathsf{Add}, 1, 3]\right\rangle\\ &\to_{\mathsf{c}} \left\langle R_0 = \#\mathsf{Add}, R_1 = 1, R_2 = 3, R_3, R_4, 3 \leftarrow \texttt{Pred}(1); 4 \leftarrow \texttt{Succ}(2); 0 \leftarrow \texttt{App}(0,\ 3);\right.\\ &\qquad\qquad \left. 0 \leftarrow \texttt{App}(0,\ 4); 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0, []\right\rangle\\ &\to_{\mathsf{c}} \langle R_0 = \#(\mathsf{Add} \mathbin{@} [0,4]), R_1 = 1, R_2 = 3, R_3 = 0, R_4 = 4, 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0, []\rangle\\ &\to_{\mathsf{c}} \langle R_0 = \#(\mathsf{Add} \mathbin{@} [0,5]), R_1 = 0, R_2 = 4, R_3 = 0, R_4 = 5, 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0, []\rangle\\ &\to_{\mathsf{c}} \#^{-1}(4).\end{aligned}$$

3.3. **Typing Extended Addressing Machines.** Recall that the set $\mathbb{T}$ of (simple) types has been introduced in Definition 2.5(1). We now show that certain EAMs can be typed, and that typable machines do not raise an error.

$$\frac{}{\mathsf{n} : \mathsf{int}}\ (\text{nat}) \qquad \frac{}{\mathsf{Y} : (\alpha \to \alpha) \to \alpha}\ (\text{fix}) \qquad \frac{R_0, \ldots, R_r \models \Delta \quad \Delta \Vdash^r (P, T) : \alpha}{\langle R_0, \ldots, R_r, P, T\rangle : \alpha}\ (\vec{R})$$

$$\frac{R_0, \ldots, R_{r-1} \models \Delta \quad !R_r = \varnothing}{R_0, \ldots, R_r \models \Delta}\ (R_\varnothing) \qquad \frac{R_0, \ldots, R_{r-1} \models \Delta \quad \#^{-1}(!R_r) : \alpha}{R_0, \ldots, R_r \models \Delta, r : \alpha}\ (R_\mathbb{T})$$

$$\frac{\Delta[i : \beta] \Vdash^r (P, []) : \alpha}{\Delta \Vdash^r (\mathtt{Load}\ i; P, []) : \beta \to \alpha}\ (\text{load}_\varnothing) \qquad \frac{\Delta[i : \beta] \Vdash^r (P, T) : \alpha \quad \#^{-1}(a) : \beta}{\Delta \Vdash^r (\mathtt{Load}\ i; P, a :: T) : \alpha}\ (\text{load}_\mathbb{T})$$

$$\frac{(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int} \Vdash^r (j \leftarrow \mathtt{Pred}(i); P, T) : \alpha}\ (\text{pred}) \qquad \frac{(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int} \Vdash^r (j \leftarrow \mathtt{Succ}(i); P, T) : \alpha}\ (\text{succ})$$

$$\frac{(\Delta, i : \mathsf{int}, j : \beta, k : \beta)[l : \beta] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int}, j : \beta, k : \beta \Vdash^r (l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T) : \alpha}\ (\text{test})$$

$$\frac{(\Delta, i : \alpha \to \beta, j : \alpha)[k : \beta] \Vdash^r (P, T) : \delta}{\Delta, i : \alpha \to \beta, j : \alpha \Vdash^r (k \leftarrow \mathtt{App}(i,\ j); P, T) : \delta}\ (\text{app})$$

$$\frac{\mathsf{M}_1 : \alpha_1 \quad \cdots \quad \mathsf{M}_n : \alpha_n}{\Delta, i : \alpha_1 \to \cdots \to \alpha_n \to \alpha \Vdash^r (\mathtt{Call}\ i, [\#\mathsf{M}_1, \ldots, \#\mathsf{M}_n]) : \alpha}\ (\text{call})$$

Figure 4: Typing rules for extended addressing machines.

Definition 3.13. (1) A *typing context* Δ is a finite set of associations between indices and types, represented as a list $i_1 : \alpha_1, \ldots, i_r : \alpha_r$. The indices $i_1, \ldots, i_r$ are not necessarily consecutive.

(2) We denote by $\Delta[i : \alpha]$ the typing context Δ where the type associated with i becomes α. Note that $\mathrm{dom}(\Delta[i : \alpha]) = \mathrm{dom}(\Delta) \cup \{i\}$. If i is not present in Δ, then $\Delta[i : \alpha] = \Delta, i : \alpha$.

(3) Let Δ be a typing environment, $\mathsf{M} \in \mathcal{M}_\mathbb{A}$, $r \geq 0$, $R_0, \ldots, R_r$ be registers, P be a program, $T \in \mathcal{T}_\mathbb{A}$ and $\alpha \in \mathbb{T}$. We define the typing judgements

$$\mathsf{M} : \alpha \qquad\qquad \Delta \Vdash^r (P, T) : \alpha \qquad\qquad R_0, \ldots, R_r \models \Delta$$

by mutual induction as the least relations closed under the rules of Figure 4.

(4) A machine M is called *typable* if the judgement $\mathsf{M} : \alpha$ is derivable for some $\alpha \in \mathbb{T}$.

The algorithm in Figure 4 deserves some discussion. As it is presented as a set of inference rules, one should reason bottom-up.

To give a machine M a type α, one needs to derive the judgement $\mathsf{M} : \alpha$. The machines n and Y are recognizable from their addresses and the rules (nat) and (fix) can thus be given higher precedence. For all other cases, one begins by giving all the registers a type using the rule $(\vec{R})$, applying $(R_\mathbb{T})$ or $(R_\varnothing)$ for each register. Once this initial step is performed, one needs to derive a judgement of the form $i_1 : \beta_{i_1}, \ldots, i_n : \beta_{i_n} \Vdash^r (P, T) : \alpha$, where P and T are the program and the input tape of the original machine respectively. This is done by verifying the coherence of the instructions in the program with the types of the registers and of the values in the input tape.

As a final consideration, notice that the rules in Figure 4 can only be considered as an algorithm when the address table map is effectively given. Otherwise, the algorithm would depend on an oracle deciding $a = \#\mathsf{M}$.

Remarks 3.14. (1) For all $\mathsf{M} \in \mathcal{M}_\mathbb{A}$ and $\alpha \in \mathbb{T}$, we have $\mathsf{M} : \alpha$ if and only if there exists $a \in \mathbb{A}$ such that both $\#^{-1}(a) : \alpha$ and $\#\mathsf{M} = a$ hold.

(2) If $\#\mathsf{M} \notin \mathbb{N} \cup \{\#\mathsf{Y}\}$, then

$$\mathsf{M} : \alpha \iff \exists \Delta \,.\, [\Delta \models \mathsf{M}.\vec{R} \ \wedge \ \Delta \Vdash^r (\mathsf{M}.P, \mathsf{M}.T) : \alpha]$$

(3) The superscript $r \geq 0$ in $\Vdash^r$ keeps track of the initial amount of registers, i.e. $\vec{i} \in \{0, \ldots, r\}$.
(4) Numeral machines are not typable by content, so (nat) having a higher priority is necessary for the consistency of the system. The (fix) rule having a higher priority does not modify the set of typable machines, but guarantees the syntax-directedness of the system.

Examples 3.15. The following are some examples of derivable typing judgements.
(1) I can be typed with $\alpha \to \alpha$ for any $\alpha \in \mathbb{T}$:

$$\dfrac{\dfrac{!R_0 = \varnothing}{R_0 \models} \quad \dfrac{\overline{0 : \mathsf{int} \Vdash^1 \langle \mathtt{Call}\ 0, [] \rangle : \alpha}}{\Vdash^1 \langle \mathtt{Load}\ 0; \mathtt{Call}\ 0, [] \rangle : \alpha \to \alpha}}{\langle R_0 = \varnothing, \mathtt{Load}\ 0; \mathtt{Call}\ 0, [] \rangle : \alpha \to \alpha}$$

(2) $\mathsf{Succ1}$ can be typed with $\mathsf{int} \to \mathsf{int}$:

$$\dfrac{\dfrac{!R_0 = \varnothing}{R_0 \models} \quad \dfrac{\dfrac{\overline{0 : \mathsf{int} \Vdash^1 \langle \mathtt{Call}\ 0, [] \rangle : \mathsf{int}}}{0 : \mathsf{int} \Vdash^1 \langle 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [] \rangle : \mathsf{int}}}{\Vdash^1 \langle \mathtt{Load}\ 0; 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [] \rangle : \mathsf{int} \to \mathsf{int}}}{\langle R_0 = \varnothing, \mathtt{Load}\ 0; 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [] \rangle : \mathsf{int} \to \mathsf{int}}$$

(3) $\mathsf{Succ2} : \mathsf{int} \to \mathsf{int}$ and $\mathsf{Add} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$ are also derivable typing judgements, but the full trees are omitted due to size constraints.

Lemma 3.16. *Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$, $\alpha \in \mathbb{T}$. Assume that $\# : \mathsf{M} \to \mathbb{A}$ is effectively given.*
(1) *If $\mathsf{M} = \langle \vec{R} = \varnothing, P, [] \rangle$ then the typing algorithm is capable of deciding whether $\mathsf{M} : \alpha$ holds.*
(2) *In general, the typing algorithm semi-decides whether $\mathsf{M} : \alpha$ holds.*

Proof. (Sketch) (i) In this case, $\mathsf{M} : \alpha$ holds if and only if $\Vdash^r (\mathsf{M}.P, [])$ does. By induction on the length of $\mathsf{M}.P$, one verifies if it is possible to construct a derivation. Otherwise, conclude that $\mathsf{M} : \alpha$ is not derivable.

(ii) In the rules $(R_{\mathbb{T}})$ and $(\mathrm{load}_{\mathbb{T}})$, one needs to show that a type for the premises exists. As the set of types is countable, and effectively given, one can easily design an algorithm constructing a derivation tree (by dovetailing). However, the algorithm cannot terminate when executed on the machine M_0 defined in Remark 3.8(2) because it would require an infinite derivation tree. □

Proposition 3.17. *Let $\mathsf{M}, \mathsf{M}', \mathsf{N}, \in \mathcal{M}_{\mathbb{A}}$ and $\alpha, \beta \in \mathbb{T}$.*
(1) *If $\mathsf{M} : \beta \to \alpha$ and $\mathsf{N} : \beta$ then $\mathsf{M} @ [\#\mathsf{N}] : \alpha$.*
(2) *If $\mathsf{M} : \alpha$ and $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$ then $\mathsf{N} : \alpha$.*
(3) *If $\mathsf{M} : \mathsf{int}$ then either M does not terminate or $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$, for some $n \geq 0$.*
(4) *If $\mathsf{M} : \alpha$ then M does not raise an error.*

Proof. (1) Simultaneously, one proves that if both $\Delta \Vdash^r (P, T) : \beta \to \alpha$ and $\mathsf{N} : \beta$ hold, then so does $\Delta \Vdash^r (P, T @ [\#\mathsf{N}]) : \alpha$. We proceed by induction on a derivation of $\mathsf{M} : \beta \to \alpha$ (resp. $\Delta \Vdash^r (P, T) : \beta \to \alpha$).

Case (nat) is vacuous.

Case (fix). By definition of Y (see Definition 3.6(2)), we have:

$$\mathsf{Y} \mathbin{@} [\#\mathsf{N}] = \left\langle R_0 = \varnothing, R_1 = \varnothing, \mathtt{Load}\ 0; 1 \leftarrow \mathtt{App}(1,\,0); 0 \leftarrow \mathtt{App}(0,\,1); \mathtt{Call}\ 0, [\#\mathsf{Y}, \#\mathsf{N}] \right\rangle.$$

Notice that, in this case, $\beta = \alpha \to \alpha$. We derive (omitting the $(R_\varnothing)$ rule usages) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0:\alpha, 1:\alpha \Vdash^2 (\mathtt{Call}\ 1, []) : \alpha}}{0:\alpha, 1:\alpha \to \alpha \Vdash^2 (1 \leftarrow \mathtt{App}(1,\,0); \ldots, []) : \alpha}}{0:(\alpha\to\alpha)\to\alpha, 1:\alpha\to\alpha \Vdash^2 (0 \leftarrow \mathtt{App}(0,\,1); \ldots, []) : \alpha} \quad \vdash \mathsf{N}:\alpha\to\alpha}{0:(\alpha\to\alpha)\to\alpha \Vdash^2 \langle \mathtt{Load}\ 1; \ldots, [\#\mathsf{N}] \rangle : \alpha} \qquad \dfrac{}{\mathsf{Y}:(\alpha\to\alpha)\to\alpha}}{\Vdash^2 \langle \mathsf{Y}.P, [\#\mathsf{Y}, \#\mathsf{N}] \rangle : \alpha}}{\mathsf{Y} \mathbin{@} [\#\mathsf{N}] \ : \alpha}$$

Case $\mathrm{load}_\varnothing$. Then $P = \mathtt{Load}\ i; P'$, $T = []$ and $\Delta[i : \beta] \Vdash^r (P', []) : \alpha$. By assumption $\mathsf{N} : \beta$, so we conclude $\Delta \Vdash^r (\mathtt{Load}\ i; P', [\#\mathsf{N}]) : \alpha$ by applying $\mathrm{load}_{\mathbb{T}}$.

All other cases derive straightforwardly from the IH.

(2) The cases $\mathsf{M} = \mathsf{Y}$ or $\mathsf{M} = \mathsf{n}$ for some $n \in \mathbb{N}$ are vacuous, as these machines are in final state. Otherwise, by Remark 3.14(2), $\Delta \Vdash^r (\mathsf{M}.P, \mathsf{M}.T) : \alpha$ for some $\Delta \models \mathsf{M}.\vec{R}$. By cases on the shape of $\mathsf{M}.P$.

Case $P = \mathtt{Load}\ i; P'$. Then $\mathsf{M}.T = a :: T'$ otherwise M would be in final state, and $\mathsf{N} = \langle \vec{R}[R_i := a], P', T' \rangle$. From $(\mathrm{Load}_{\mathbb{T}})$ we get $\Delta[i : \beta] \Vdash^r (P', T') : \alpha$ for some $\beta \in \mathbb{T}$ satisfying $\#^{-1}(a) : \beta$. As $\vec{R} \models \Delta$ we derive $\vec{R}[R_i := a] \models \Delta[i : \beta]$, so as $N = \left\langle \vec{R}[R_i := a], P', T' \right\rangle$, by Remark 3.14(2), $N : \alpha$.

Case $P = \mathtt{Call}\ i$. Then $i : \alpha_1 \to \cdots \to \alpha_n \to \alpha$, $T = [\#\mathsf{M}_1, \ldots, \#\mathsf{M}_n]$ and $\mathsf{M}_j : \alpha_j$, for all $j \leq n$. In this case, $\mathsf{N} = \#^{-1}(!(\mathsf{M}.R_i)) \mathbin{@} T$ with $\#^{-1}(!(\mathsf{M}.R_i)) : \alpha_1 \to \cdots \to \alpha_n \to \alpha$, so we conclude by (1).

All other cases follows easily from the IH.

(3) Assume that $\mathsf{M} : \mathsf{int}$ and $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ for some N in final state. By (2), we obtain that $\mathsf{N} : \mathsf{int}$ holds, thus $\mathsf{N} = \mathsf{n}$ since numerals are the only machines in final state typable with int.

(4) The three cases from Figure 3 where a machine can raise an error are ruled out by the typing rules (pred), (succ) and (test), respectively. Therefore, no error can be raised during the execution. □

4. Simulating (E)PCF

In this section we define a translation from EPCF terms to EAMs. We prove that both the typing and the operational semantics of EPCF programs are preserved under this translation. In Theorem 2.23, we show that a PCF program having type int computes a numeral $\underline{n}$ exactly when the corresponding EAM reduces to n. This result builds upon [IMM23, Thm. 4.12], where only one implication is shown. It can be seen as a first step towards full abstraction.

4.1. **EAMs implementing PCF instructions.** We start by defining some auxiliary EAMs implementing the main instructions of PCF.

Definition 4.1. Define the following EAMs (for $k > 0$ and $n \geq 0$):

$$\begin{aligned}
\mathsf{Pr}_i^k &= \left\langle R_0, (\texttt{Load 1})^{i-1}; \texttt{Load 0}; (\texttt{Load 1})^{k-i}; \texttt{Call 0}; [] \right\rangle, \text{ for } 1 \leq i \leq k;\\
\mathsf{Pred} &= \left\langle R_0, \texttt{Load 0}; 0 \leftarrow \texttt{Pred}(0); \texttt{Call 0}; [] \right\rangle;\\
\mathsf{Succ} &= \left\langle R_0, \texttt{Load 0}; 0 \leftarrow \texttt{Succ}(0); \texttt{Call 0}; [] \right\rangle;\\
\\
\mathsf{Ifz} &= \left\langle R_0, R_1, R_2, \texttt{Load 0}; \texttt{Load 1}; \texttt{Load 2}; 0 \leftarrow \texttt{Test}(0,\, 1,\, 2); \texttt{Call 0}; [] \right\rangle;\\
\mathsf{Apply}_0^k &= \mathsf{Pr}_1^1, \text{ for } k > 0. \text{ Recall that the EAM } \mathsf{Pr}_1^1 = \mathsf{I} \text{ represents the identity;}\\
\mathsf{Apply}_{n+1}^k &= \left\langle \begin{array}{c} R_0 = \#\mathsf{Apply}_n^k, R_1, \ldots, R_{k+2}, \texttt{Load } (1, \ldots, k+2);\\ 2 \leftarrow \texttt{App}(2,\, k+2); \cdots; k+1 \leftarrow \texttt{App}(k+1,\, k+2);\\ 0 \leftarrow \texttt{App}(0,\, 1); \cdots; 0 \leftarrow \texttt{App}(0,\, k+1); \texttt{Call } 0, [] \end{array} \right\rangle.
\end{aligned}$$

The registers whose values are not specified are assumed to be uninitialized.

The EAM Apply_n^k deserves an explanation. This machine is stuck, waiting for $n+k+1$ arguments $a, d_1, \ldots, d_k, e_1, \ldots, e_n$, where k is the arity of a and n is the arity of each d_i. Once all arguments are available in the tape, Apply_n^k applies $\vec{e}$ to each d_i and then feeds the machine $\#^{-1}(a)$ the resulting list of arguments $d_1 \cdot \vec{e}, \ldots, d_k \cdot \vec{e}$.

Lemma 4.2. *For all $a, b, c, d_1, \ldots, d_k, e_1, \ldots, e_n \in \mathbb{A}$, the EAMs below reduce as follows:*

(1) $\mathsf{Pr}_i^k \mathbin{@} [d_1, \ldots, d_k] \twoheadrightarrow_{\mathsf{c}}^{k+1} d_i$, *for* $1 \leq i \leq k > 0$;
(2) $\mathsf{Apply}_n^k \mathbin{@} [a, d_1, \ldots, d_k, e_1, \ldots, e_n] \twoheadrightarrow_{\mathsf{c}}^{(3k+4)n+2} \#^{-1}(a) \mathbin{@} [d_1 \cdot e_1 \cdots e_n, \ldots, d_k \cdot e_1 \cdots e_n]$;
(3) $\mathsf{Pred} \mathbin{@} [a] \to_{\mathsf{c}} \left\langle R_0 = a, 0 \leftarrow \texttt{Pred}(0); \texttt{Call } 0, [] \right\rangle$;
(4) $\mathsf{Succ} \mathbin{@} [a] \to_{\mathsf{c}} \left\langle R_0 = a, 0 \leftarrow \texttt{Succ}(0); \texttt{Call } 0, [] \right\rangle$;
(5) $\mathsf{Ifz} \mathbin{@} [a, b, c] \twoheadrightarrow_{\mathsf{c}}^{3} \left\langle R_0 = a, R_1 = b, R_2 = c, 0 \leftarrow \texttt{Test}(0,\, 1,\, 2); \texttt{Call } 0, [] \right\rangle$;
(6) $\mathsf{Y} \mathbin{@} [a] \twoheadrightarrow_{\mathsf{c}}^{5} \#^{-1}(a) \mathbin{@} [\#(\mathsf{Y} \mathbin{@} [a]\,)]$.

Proof. The only interesting cases are the application and fixed point combinator.

(2) Concerning Apply_n^k, we proceed by induction on n.

- Base case. If $n = 0$ then $\mathsf{Apply}_0^k = \mathsf{I}$, and $\mathsf{I} \mathbin{@} [a, d_1, \ldots, d_k] \twoheadrightarrow_{\mathsf{c}}^{2} \#^{-1}(a) \mathbin{@} [d_1, \ldots, d_k]$.
- Induction case. Easy calculations give:

$$\mathsf{Apply}_{n+1}^k \mathbin{@} [a, d_1, \ldots, d_k, e_1, \ldots, e_{n+1}] \twoheadrightarrow_{\mathsf{c}}^{3k+4} \mathsf{Apply}_n^k \mathbin{@} [a, d_1 \cdot e_1, \ldots, d_k \cdot e_1, e_2, \ldots, e_{n+1}]$$

This case follows from the IH since $3k + 4 + (3k+4)n + 2 = (3k+4)(n+1) + 2$.

(6) Recall that Y has been introduced in Definitions 3.5(6) and 3.6(2).

$$\begin{aligned}
\mathsf{Y} &= \langle R_0, R_1, \texttt{Load } (0,1); 0 \leftarrow \texttt{App}(0,\, 1); 1 \leftarrow \texttt{App}(1,\, 0); \texttt{Call } 1, [\#\mathsf{Y}, a] \rangle\\
&\twoheadrightarrow_{\mathsf{c}}^{2} \langle R_0 = \#\mathsf{Y}, R_1 = a, 0 \leftarrow \texttt{App}(0,\, 1); 1 \leftarrow \texttt{App}(1,\, 0); \texttt{Call } 1, [] \rangle\\
&\to_{\mathsf{c}} \langle R_0 = \#\mathsf{Y} \cdot a, R_1 = a, 1 \leftarrow \texttt{App}(1,\, 0); \texttt{Call } 1, [] \rangle\\
&\to_{\mathsf{c}} \langle R_0 = \#\mathsf{Y} \cdot a, R_1 = a \cdot (\#\mathsf{Y} \cdot a), \texttt{Call } 1, [] \rangle\\
&\to_{\mathsf{c}} \#^{-1}(a \cdot (\#\mathsf{Y} \cdot a)) = \#^{-1}(a) \mathbin{@} [\#\mathsf{Y} \mathbin{@} [a]\,].
\end{aligned}$$

This concludes the proof. □

We show that the above machines are actually typable using the type assignment system of Figure 4. This property is needed to show that the translation is type-preserving.

Lemma 4.3. *The EAMs introduced in Definition 4.1 can be typed as follows (for all $\alpha, \beta_i, \delta_i \in \mathbb{T}$, using the notation $\vec{\delta} \to \beta_i = \delta_1 \to \cdots \to \delta_n \to \beta_i$):*

(1) $\mathsf{Pr}_i^k : \beta_1 \to \cdots \to \beta_k \to \beta_i$, *for* $1 \leq i \leq k > 0$;
(2) $\mathsf{Apply}_n^k : (\beta_1 \to \cdots \to \beta_k \to \alpha) \to (\vec{\delta} \to \beta_1) \to \cdots \to (\vec{\delta} \to \beta_k) \to \vec{\delta} \to \alpha$;
(3) $\mathsf{Pred} : \mathsf{int} \to \mathsf{int}$;
(4) $\mathsf{Succ} : \mathsf{int} \to \mathsf{int}$;
(5) $\mathsf{Ifz} : \mathsf{int} \to \alpha \to \alpha \to \alpha$.

Proof. It follows easily from Definition 4.1. For Apply_n^k, proceed by induction on n. □

4.2. **Translating (E)PCF programs to EAMs.** Using the auxiliary EAM introduced above, we can translate an EPCF term M having $x_1, \ldots, x_n$ as free variables as an EAM M loading their values from the input tape.

Definition 4.4 (Translation)**.** Let M be an EPCF term such that $\mathrm{FV}(M) \subseteq \{x_1, \ldots, x_n\}$. The *translation* of M (w.r.t. $\vec{x}$) is an EAM $|M|_{\vec{x}}$ defined by structural induction on M:

$$
\begin{array}{lcl}
|x_i|_{\vec{x}} & = & \mathsf{Pr}_i^n, \text{ where } i \in \{1, \ldots, n\}; \\
|\lambda y.M|_{\vec{x}} & = & |M|_{\vec{x},y}, \text{ where wlog } y \notin \vec{x}; \\
|M\langle N/y\rangle|_{\vec{x}} & = & |M|_{y,\vec{x}} \mathbin{@} [\#|N|]\,; \\
|M \cdot N|_{\vec{x}} & = & \mathsf{Apply}_n^2 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}]\,; \\
|\mathbf{0}|_{\vec{x}} & = & \mathsf{Pr}_1^{n+1} \mathbin{@} [0]\,; \\
|\mathbf{pred}\, M|_{\vec{x}} & = & \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Pred}, \#|M|_{\vec{x}}]\,; \\
|\mathbf{succ}\, M|_{\vec{x}} & = & \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Succ}, \#|M|_{\vec{x}}]\,; \\
|\mathbf{ifz}(L, M, N)|_{\vec{x}} & = & \mathsf{Apply}_n^3 \mathbin{@} [\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}]\,; \\
|\mathbf{fix}\, M|_{\vec{x}} & = & \begin{cases} \mathsf{Y} \mathbin{@} [\#|M|_{\vec{x}}]\,, & \text{if } n = 0, \\ \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Y}, \#|M|_{\vec{x}}]\,, & \text{otherwise.} \end{cases}
\end{array}
$$

Examples 4.5. Recall the EPCF terms introduced in Example 2.4. The translation of said EPCF terms produces the following machines:

(1) $|\mathbf{I}| = |x|_x = \mathsf{Pr}_1^1$.
(2) $|\mathbf{\Omega}| = \mathsf{Y} \mathbin{@} [\#|\mathbf{I}|] = \mathsf{Y} \mathbin{@} [\#\mathsf{Pr}_1^1]$.
(3) $|\mathbf{succ1}| = |\lambda x.\mathbf{succ}\, x| = |\mathbf{succ}\, x|_x = \mathsf{Apply}_1^1 \mathbin{@} [\#\mathsf{Succ}, \#\mathsf{Pr}_1^1]$,
(4)
$$
\begin{array}{rcl}
|\mathbf{succ2}| & = & \mathsf{Apply}_2^0 \mathbin{@} [\#|s \cdot (s \cdot n)|_{s,n} \cdot \#|\mathbf{succ1}|] \\
& = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot \#|s \cdot n|_{s,n}, \#|\mathbf{succ1}|] \\
& = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot (\#\mathsf{Apply}_2^2 \cdot \mathsf{Pr}_1^2 \cdot \mathsf{Pr}_2^2), \#|\mathbf{succ1}|] \\
& = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot (\#\mathsf{Apply}_2^2 \cdot \mathsf{Pr}_1^2 \cdot \mathsf{Pr}_2^2), \#\mathsf{Apply}_1^1 \cdot \#\mathsf{Succ} \cdot \#\mathsf{Pr}_1^1]\,.
\end{array}
$$

In the translation above typing information was deliberately ignored for the sake of generality. We now show that our translation preserves the typings in the following sense.

Theorem 4.6. *Let M be an* (E)PCF *term and $\Gamma = x_1 : \delta_1, \ldots, x_n : \delta_n$. Then*

$$\Gamma \vdash M : \alpha \;\Rightarrow\; |M|_{\vec{x}} : \delta_1 \to \cdots \to \delta_n \to \alpha.$$

Proof (Appendix A.2). Note that the type assignment systems of EPCF and of PCF coincide on PCF terms. Proceed by induction on a derivation of $\Gamma \vdash M : \alpha$, using Proposition 3.17(1) and Lemma 4.3. □

Ideally, one would like that an EPCF step $M \to_{\mathtt{wh}} N$ becomes a reduction $|M| \twoheadrightarrow_{\mathsf{c}} |N|$ in the corresponding EAMs. Unfortunately, the situation is more complicated—the translation $|N|$ may contain auxiliary EAMs that are not generated by $|M|$ along reduction. The property that actually holds is that the two EAMs are interconvertible $|M| \leftrightarrow_{\mathsf{c}} |N|$, and $|N|$ is 'closer' to their common reduct. The next definition captures this intuition.

Definition 4.7. For $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$, define the relation:

$$\mathsf{M} \succ_{\mathsf{c}} \mathsf{N} \iff \exists \mathsf{Z} \in \mathcal{M}_{\mathbb{A}} \,.\, [\, (\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z} \;{}_{\mathsf{c}}\!\!\twoheadleftarrow \mathsf{N}) \wedge (|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}| > |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}|) \,].$$

Note that the reduction path $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}$ above must be non-empty. Moreover, recall that $\mathsf{M} \succ_{\mathsf{c}} \mathsf{N}$ entails $\mathsf{M} \leftrightarrow_{\mathsf{c}} \mathsf{N}$.

Lemma 4.8. (1) $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}| > |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}|$ *and* $\mathsf{Z} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'$ *imply* $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'| > |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'|$.
(2) *The relation* $\succ_{\mathsf{c}}$ *is transitive.*

Proof. (1) By confluence and determinism of $\to_{\mathsf{c}}$ (Lemma 3.11).
(2) follows from (1). □

Proposition 4.9. *Given* EPCF *program* M *of type* int*, we have* $M \to_{\mathtt{wh}} N \Rightarrow |M| \succ_{\mathsf{c}} |N|$.

Proof (Appendix A.2). By induction on a derivation of $M \to_{\mathtt{wh}} N$, applying Lemma 4.2. □

Since every PCF program P of type int is also an EPCF program of the same type, and in this case the operational semantics of PCF and EPCF coincide (Theorem 2.23), we can use the above proposition to prove that the EAM $|P|$ faithfully simulates the behavior of P.

Theorem 4.10. *For a* PCF *program* P *having type* int*, the following are equivalent:*

(1) $P \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$;
(2) $|P| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$.

Proof. (1 ⇒ 2) Let $P \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. Equivalently, $P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ (by Theorem 2.23). By Proposition 4.9, we get $|P| \leftrightarrow_{\mathsf{c}} \mathsf{n}$. Since n is in final state, this entails $|P| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$ by confluence (Lemma 3.11(2)).

(2 ⇒ 1) We prove the contrapositive. Assume that $\vdash P : \mathsf{int}$, but P does not reduce to a numeral. As PCF enjoys subject reduction [Ong95], P must have an infinite $\to_{\mathsf{PCF}}$ reduction path. By Theorem 2.23, this generates an infinite w.h. reduction path $P = P_0 \to_{\mathtt{wh}} P_1 \twoheadrightarrow_{\mathtt{wh}} P_k \twoheadrightarrow_{\mathtt{wh}} \cdots$. By Proposition 4.9, this translates to an infinite $\succ_{\mathsf{c}}$-chain $|P_k| \succ_{\mathsf{c}} |P_{k+1}|$ for common reducts $|P_k| \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}_k \;{}_{\mathsf{c}}\!\!\twoheadleftarrow |P_{k+1}|$. By confluence and Lemma 4.8(1), there are Z'_k and $i_k > j_k$ such that

$$\begin{array}{rl} & |P_1| \;\overset{j_0}{\searrow}\; |P_2| \;\overset{j_1}{\searrow}\; |P_3| \;\overset{j_2}{\searrow}\; \cdots \\ |P| = |P_0| \twoheadrightarrow_{\mathsf{c}}^{i_0} \mathsf{Z}'_1 \twoheadrightarrow_{\mathsf{c}}^{i_1} \mathsf{Z}'_2 \twoheadrightarrow_{\mathsf{c}}^{i_2} \mathsf{Z}'_3 \twoheadrightarrow_{\mathsf{c}} \cdots & \end{array}$$

This is only possible if the EAM $|P|$ does not terminate. □

5. A Fully Abstract Computational Model

We construct a model of PCF based on EAMs and prove that it is fully abstract. The main technical tools used to achieve this result are logical relations [AC98, Pit97] and definability [Cur07]. We remind the definition of a fully abstract model of PCF, together with some basic results about its observational equivalence.

Recall that PCF contexts have been defined in Definition 2.2(1), the relation $\to_{\mathsf{PCF}}$ represents a weak head reduction step and generates the interconvertibility relation $\leftrightarrow_{\mathsf{PCF}}$.

Definition 5.1. (1) Given a PCF context $\mathrm{C}\Box$ an environment Γ and two types $\alpha, \beta \in \mathbb{T}$, we write $\mathrm{C}\Box : (\Gamma, \alpha) \to \beta$ if $\vdash \mathrm{C}[P] : \beta$ holds, for all PCF terms P having type α in Γ, i.e. satisfying $\Gamma \vdash P : \alpha$.

(2) The *observational equivalence* is defined on PCF terms P, P' having type α in Γ by:

$$P \equiv_{\mathrm{obs}} P' \iff \forall \mathrm{C}\Box : (\Gamma, \alpha) \to \mathsf{int}\,.\,[\,\mathrm{C}[P] \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff \mathrm{C}[P'] \twoheadrightarrow_{\mathsf{PCF}} \underline{n}\,]$$

(3) The *applicative equivalence* is defined on PCF programs P, P' having the same type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{int}$, for some $k \geq 0$, as follows:

$$P \equiv_{\mathrm{app}} P' \iff \forall Q_1 : \alpha_1, \ldots, Q_k : \alpha_k\,.\,[\,PQ_1 \cdots Q_k \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff P'Q_1 \cdots Q_k \twoheadrightarrow_{\mathsf{PCF}} \underline{n}\,]$$

Remark 5.2. Note that PCF contexts are different from PCF evaluation contexts—unlike evaluation contexts, the 'hole' can be found at any location in the term.

The main difference between observational equivalence in the untyped setting and the one for PCF is that, in the latter, we observe termination at ground type. The following result shows that applicative and observational equivalence coincide on closed PCF terms.

Proposition 5.3 Abramsky's equivalence [Ong95]**.**
For all PCF *programs* P, P' *of type* α*, we have:*

$$P \equiv_{\mathrm{obs}} P' \iff P \equiv_{\mathrm{app}} P'$$

Definition 5.4. A *model of* PCF is a triple $\mathscr{M} = \langle (\mathcal{M}_\alpha)_{\alpha \in \mathbb{T}}, (\ \cdot^{\alpha,\beta})_{\alpha,\beta \in \mathbb{T}}, [\![-]\!] \rangle$ where:

- $(\mathcal{M}_\alpha)_{\alpha \in \mathbb{T}}$ is a type-indexed collection of sets;
- $(\ \cdot^{\alpha,\beta}) : \mathcal{M}_{\alpha \to \beta} \times \mathcal{M}_\alpha \to \mathcal{M}_\beta$ is a well-typed operation called *application*;
- $[\![-]\!]$ is an *interpretation function* mapping a derivation of $x_1 : \beta_1, \ldots, x_n : \beta_n \vdash P : \alpha$ to an element $[\![x_1 : \beta_1, \ldots, x_n : \beta_n \vdash P : \alpha]\!] \in \mathcal{M}_{\beta_1 \to \cdots \to \beta_n \to \alpha}$.

Due to syntax-directedness of PCF type system, we can simply write $[\![P]\!]^{\Gamma,\alpha}$ for $[\![\Gamma \vdash P : \alpha]\!]$.

Definition 5.5. (1) A model $\mathscr{M}$ is *sound* if, for all PCF programs P, P' of type α, the following holds:

$$P \leftrightarrow_{\mathsf{PCF}} P' \Rightarrow [\![P]\!]^\alpha = [\![P']\!]^\alpha$$

(2) A model $\mathscr{M}$ of PCF is *fully abstract* if, for all PCF programs P, P' of type α:

$$[\![P]\!]^\alpha = [\![P']\!]^\alpha \iff P \equiv_{\mathrm{obs}} P'$$

The left-to-right implication is called *adequacy*, the converse implication *completeness*.

5.1. **An adequate model of PCF.** In this section we are going to construct a model of PCF based on EAMs, and we prove that it enjoys adequacy. The idea is to focus on the subset $\mathcal{D} \subseteq \mathbb{A}$ containing addresses of typable EAMs, which is then stratified $(\mathcal{D}_\alpha)_{\alpha\in\mathbb{T}}$ following the inductive syntax of simple types. In other words, the set $\mathcal{D}_\alpha$ contains the addresses of those EAM having type α. In particular, this means that the sets $\mathcal{D}_\alpha$ are not pairwise disjoint: e.g., the address $\#\mathsf{I}$ of the identity machine belongs to the set $\mathcal{D}_{\alpha\to\alpha}$, for all types α. The well-typedness condition allows us to get rid of those EAMs containing addresses of infinite chains of 'pointers'—a phenomenon described in Remark 3.8(2)—that might exist in $\mathcal{M}_\mathbb{A}$, but cannot be typed. Subsequently, we are going to impose that two addresses $a, b \in \mathcal{D}_\alpha$ are equal in the model whenever the corresponding EAMs exhibit the same applicative behavior: at ground type ($\alpha = \mathsf{int}$) this simply means that either $\#^{-1}(a)$ and $\#^{-1}(b)$ compute the same numeral, or they are both non-terminating. This equality is then lifted at higher order types following the well-established tradition of logical relations [AC98, Pit97].

The above intuitions are formalized in the following definition.

Definition 5.6. (1) For all types $\alpha \in \mathbb{T}$, define $\mathcal{D}_\alpha = \{a \in \mathbb{A} \mid \#^{-1}(a) : \alpha\}$.
(2) For all EAMs M, N of type α, define $\mathsf{M} \equiv_\alpha \mathsf{N}$ by structural induction on α:

$$\mathsf{M} \equiv_{\mathsf{int}} \mathsf{N} \quad \iff \quad \forall n \in \mathbb{N}\,.\,\big[\,\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \iff \mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{n}\,\big]$$
$$\mathsf{M} \equiv_{\alpha\to\beta} \mathsf{N} \quad \iff \quad \forall a, b \in \mathcal{D}_\alpha\,.\,\big[\,\#^{-1}(a) \equiv_\alpha \#^{-1}(b) \;\Rightarrow\; \mathsf{M}\,@\,[a] \equiv_\beta \mathsf{N}\,@\,[b]\,\big]$$

(3) For $\alpha \in \mathbb{T}$ and $a, b \in \mathcal{D}_\alpha$, we write $a \simeq_\alpha b$ whenever $\#^{-1}(a) \equiv_\alpha \#^{-1}(b)$ holds.
(4) We let $\mathcal{D}_\alpha/_{\simeq_\alpha} = \{[a]_{\simeq_\alpha} \mid a \in \mathcal{D}_\alpha\}$.

The model which is shown to be fully abstract is constructed as follows.

Definition 5.7. Define the model $\mathscr{D} = \langle (\mathcal{D}_\alpha/_{\simeq_\alpha})_{\alpha\in\mathbb{T}}, (\;\cdot^{\alpha,\beta})_{\alpha,\beta\in\mathbb{T}}, [\![-]\!]\rangle$ where

$$\begin{array}{rcl} [a]_{\simeq_{\alpha\to\beta}} \cdot^{\alpha,\beta} [b]_{\simeq_\alpha} & = & [a \cdot b]_{\simeq_\beta} \\ [\![x_1 : \beta_1, \ldots, x_n : \beta_n \vdash P : \alpha]\!] & = & [\#|P|_{\vec{x}}]_{\simeq_{\beta_1\to\cdots\to\beta_n\to\alpha}} \end{array}$$

The application $\cdot^{\alpha,\beta}$ is well defined by Proposition 3.17(1) and the interpretation function $[\![-]\!]$ by Theorem 4.6. Note that two PCF programs P, Q of type α have the same interpretation in the model $\mathscr{D}$, i.e. $[\![P]\!]^\alpha = [\![Q]\!]^\alpha$, exactly when $|P| \simeq_\alpha |Q|$. Hence, we mainly work with translations of PCF terms modulo $\simeq_\alpha$ and draw conclusions for $\mathscr{D}$ at the end (Theorem 5.20).

Lemma 5.8. (1) *The relation $\equiv_\alpha$ is an equivalence.*
(2) *Let $\mathsf{M}, \mathsf{N} \in \mathcal{M}_\mathbb{A}$ and $\alpha \in \mathbb{T}$ be such that $\mathsf{M} : \alpha$ and $\mathsf{N} : \alpha$. If $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$ then $\mathsf{M} \equiv_\alpha \mathsf{N}$.*
(3) *Assume $\mathsf{M} : \alpha \to \beta, \mathsf{N}_1 : \alpha$ and $\mathsf{N}_2 : \alpha$. If $\mathsf{N}_1 \to_{\mathsf{c}} \mathsf{N}_2$ then $\mathsf{M}\,@\,[\#\mathsf{N}_1] \equiv_\beta \mathsf{M}\,@\,[\#\mathsf{N}_2]$.*

Proof. (1) Reflexivity, symmetry, and transitivity are proven easily by induction on α.
(2) By induction on α, using the reflexivity and transitivity properties from (1).
(3) By (2) we get $\mathsf{N}_1 \equiv_\beta \mathsf{N}_2$, so this item follows from reflexivity and definition of $\equiv_{\beta\to\alpha}$. □

Everything is now in place to prove that the model is sound and adequate.

Corollary 5.9 (Soundness)**.** *For all PCF programs P_1, P_2 of type α, we have*

$$|P_1| \leftrightarrow_{\mathsf{c}} |P_2| \;\Rightarrow\; |P_1| \equiv_\alpha |P_2|$$

Proof. By definition of interconvertibility $\leftrightarrow_{\mathsf{c}}$ and Lemma 5.8(2). □

Theorem 5.10 (Adequacy)**.** *Given two PCF programs P_1, P_2 of type α, we have*

$$|P_1| \equiv_\alpha |P_2| \;\Rightarrow\; P_1 \equiv_{\mathrm{obs}} P_2.$$

Proof. By Proposition 5.3 it is sufficient to show $P_1 \equiv_{\mathrm{app}} P_2$. Proceed by induction on α.

Base case $\alpha = \mathsf{int}$. For $i = 1, 2$, we have $P_i \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff |P_i| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$, by Theorem 4.10. Then, this case follows from the assumption $|P_1| \equiv_{\mathsf{int}} |P_2|$, that is $|P_1| \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \iff |P_2| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$.

Case $\alpha = \beta_1 \to \beta_2$. Take any PCF program $\vdash Q_1 : \beta_1$. By Definitions 4.4 and 4.1, we get $|P_1Q_1| = \mathsf{Pr}^1_1 \mathbin{@} [\#\mathsf{Pr}^1_1, \#|P_1|, \#|Q_1|] \twoheadrightarrow_{\mathsf{c}} |P_1| \mathbin{@} [\#|Q_1|]$ whence $|P_1Q_1| \equiv_{\beta_2} |P_1| \mathbin{@} [\#|Q_1|]$. Similarly, $|P_2Q_1| \equiv_{\beta_2} |P_2| \mathbin{@} [\#|Q_1|]$. From $|P_1| \equiv_{\beta_1 \to \beta_2} |P_2|$ and $|Q_1| \equiv_{\beta_1} |Q_1|$ (reflexivity), we get $|P_1| \mathbin{@} [\#|Q_1|] \equiv_{\beta_2} |P_2| \mathbin{@} [\#|Q_1|]$. By transitivity of $\equiv_{\beta_2}$, we get $|P_1Q_1| \equiv_{\beta_2} |P_2Q_1|$ and by IH $P_1Q_1 \equiv_{\mathrm{app}} P_2Q_1$. Thus, for all $Q_2, \dots, Q_k$ of the appropriate types, we get $P_1Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff P_2Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. As Q_1 is arbitrary, conclude $P_1 \equiv_{\mathrm{app}} P_2$. □

The following technical lemmas turn out to be useful in the remainder of the article.

Lemma 5.11. *For* (E)PCF *programs* $M, N_1, \dots, N_n$ *such that* $\vdash M : \alpha_1 \to \cdots \to \alpha_n \to \beta$ *and* $\vdash N_i : \alpha_i$ *for all* $i\,(1 \le i \le n)$*, we have*

$$|M| \mathbin{@} [\#|N_1|, \dots, \#|N_n|] \equiv_\beta |M \cdot N_1 \cdots N_n|.$$

Proof. By induction on n, the case $n = 0$ being trivial.

Case $n > 0$. By Definitions 4.4 and 4.1, we get

$$|MN_1 \cdots N_n| = \mathsf{Pr}^1_1 \mathbin{@} [\#\mathsf{Pr}^1_1, \#|MN_1 \cdots N_{n-1}|, \#|N_n|] \twoheadrightarrow_{\mathsf{c}} |MN_1 \cdots N_{n-1}| \mathbin{@} [\#|N_n|],$$

whence $|MN_1 \cdots N_n| \equiv_\beta |MN_1 \cdots N_{n-1}| \mathbin{@} [\#|N_n|]$ by Lemma 5.8(2). By IH, we have $|M| \mathbin{@} [\#|N_1|, \dots, \#|N_{n-1}|] \equiv_\beta |MN_1 \cdots N_{n-1}|$, so we conclude by Lemma 5.8(1). □

Lemma 5.12 (Strengthening for EAMs)**.**
Let $M \in \Lambda^{\mathsf{E}}$ *be such that* $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash M : \beta$*. Assume that* $x_i \notin \mathrm{FV}(M)$ *for some* $i\,(1 \le i \le n)$*. Then for all* $a_1 \in \mathcal{D}_{\alpha_1}, \dots, a_n \in \mathcal{D}_{\alpha_n}$*:*

$$|M|_{x_1,\dots,x_n} \mathbin{@} [a_1, \dots, a_n] \equiv_\beta |M|_{x_1,\dots,x_{i-1},x_{i+1},\dots,x_n} \mathbin{@} [a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n]$$

Proof (Appendix A.3). By induction on a derivation of $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash M : \beta$. □

Corollary 5.13. *Let* $M \in \mathcal{P}_{\mathsf{E}}$ *and* $\alpha_1, \dots, \alpha_n, \beta \in \mathbb{T}$*. If* $\vdash M : \beta$ *then, for all* $x_1, \dots, x_n \in \mathrm{Var}$ *and* $a_i \in \mathcal{D}_{\alpha_i}$ $(1 \le i \le n)$*, we have*

$$|M|_{x_1,\dots,x_n} \mathbin{@} [a_1, \dots, a_n] \equiv_\beta |M|\,.$$

5.2. **Definability and full abstraction.** The adequacy result established above gives one implication of the Full Abstraction property. In order to prove the converse implication, namely completeness, we need to show that the model does not contain any undefinable element (*junk*). This amounts to associate with any typable EAM, a PCF program of the same type exhibiting the same observational behavior (*mutatis mutandi*). The problem is that an EAM might perform useless computations (e.g., updating the value of a register that is subsequently never used) that could however prevent the machine from terminating. To simulate the same behavior in the corresponding PCF term, we use a 'convergency' test $\mathtt{Ifc}(x, y)$ (*if-converges-then*) defined by: $\mathtt{Ifc}(x, y) = \mathbf{ifz}(x, y, y)$. Indeed, given PCF programs P and Q, the program $\mathtt{Ifc}(P, Q)$ is observationally indistinguishable from Q exactly when P is terminating, independently from its result.

Recall that an EAM typing context $\Delta = i_1 : \beta_{i_1}, \dots, i_k : \beta_{i_k}$ is a list of associations between indices of registers and types. Moreover, the judgments $\mathsf{M} : \alpha$, $\Delta \Vdash^r (P, T) : \alpha$ and $\vec{R} \models \Delta$ have been defined in Figure 4.

Definition 5.14 (Reverse Translation)**.** Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$, P be an EAM program, $T \in \mathcal{T}_{\mathbb{A}}$, $\alpha \in \mathbb{T}$. Given $\mathsf{M} : \alpha$ (resp. $\Delta \Vdash^r (P,T) : \alpha$), we associate a PCF term $(\!|\mathsf{M}|\!)_\alpha$ (resp. $(\!|P,T|\!)^\Delta_\alpha$) defined by induction on the type-derivation as follows:

$(\!|\mathsf{n}|\!)_{\mathsf{int}} = \underline{n}$;

$(\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha} = \lambda x.\mathbf{fix}\, x$;

$(\!|\langle \vec{R}, P, T\rangle|\!)_\alpha = (\lambda x_{i_1} \dots x_{i_k}.(\!|P,T|\!)^{i_1:\beta_{i_1},\dots,i_k:\beta_{i_k}}_\alpha) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_k})|\!)_{\beta_{i_k}}$,
where $\vec{R} \models i_1 : \beta_{i_1}, \dots, i_k : \beta_{i_k}$, for $1 \le k \le r$;

$(\!|\mathtt{Load}\ i; P, []|\!)^\Delta_{\beta\to\alpha} = \lambda x_i.(\!|P, []|\!)^{\Delta[i:\beta]}_\alpha$;

$(\!|\mathtt{Load}\ i; P, a :: T|\!)^\Delta_\alpha = \big(\lambda x_i.(\!|P,T|\!)^{\Delta[i:\beta]}_\alpha\big) \cdot (\!|\#^{-1}(a)|\!)_\beta$;

$(\!|j \leftarrow \mathtt{Pred}(i); P, T|\!)^{\Delta,i:\mathsf{int}}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_j.(\!|P,T|\!)^{(\Delta,i:\mathsf{int})[j:\mathsf{int}]}_\alpha) \cdot (\mathbf{pred}\, x_i))$;

$(\!|j \leftarrow \mathtt{Succ}(i); P, T|\!)^{\Delta,i:\mathsf{int}}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_j.(\!|P,T|\!)^{(\Delta,i:\mathsf{int})[j:\mathsf{int}]}_\alpha) \cdot (\mathbf{succ}\, x_i))$;

$(\!|l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T|\!)^{\Delta,i:\mathsf{int},j:\beta,k:\beta}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_l.(\!|P,T|\!)^{(\Delta,i:\mathsf{int},j:\beta,k:\beta)[l:\beta]}_\alpha) \cdot \mathbf{ifz}(x_i, x_j, x_k))$;

$(\!|k \leftarrow \mathtt{App}(i,\ j); P, T|\!)^{\Delta,i:\beta\to\alpha,j:\beta}_\gamma = (\lambda x_k.(\!|P,T|\!)^{(\Delta,i:\beta\to\alpha,j:\beta)[k:\alpha]}_\gamma) \cdot (x_i \cdot x_j)$;

$(\!|\mathtt{Call}\ i, [a_1, \dots, a_n]|\!)^{\Delta,i:\alpha_1\to\dots\to\alpha_n\to\alpha}_\alpha = x_i \cdot (\!|\#^{-1}(a_1)|\!)_{\alpha_1} \cdots (\!|\#^{-1}(a_n)|\!)_{\alpha_n}$.

It is easy to check that the PCF term associated with $\mathsf{M} : \alpha$ is actually a PCF program.

Examples 5.15. Consider the EAMs introduced in Example 3.9. The reverse translation applied to said machines produces the following PCF programs:

(1) $(\!|\mathsf{I}|\!)_{\alpha\to\alpha} = (\!|\mathtt{Load}\ 0; \mathtt{Call}\ 0, []|\!)_{\alpha\to\alpha} = \lambda x_0.(\!|\mathtt{Call}\ 0, []|\!)^{0:\alpha}_\alpha = \lambda x_0.x_0$.

(2) $(\!|\mathsf{Y} \mathbin{@} [\#\mathsf{I}]|\!)_\alpha = (\!|\mathtt{Load}\ 0; \mathtt{Load}\ 1; 0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1; [\#\mathsf{Y}, \#\mathsf{I}]|\!)_\alpha$
$= (\lambda x_0.(\lambda x_1.(\!|0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1; []|\!)_\alpha) \cdot (\!|\mathsf{I}|\!)_{\alpha\to\alpha}) \cdot (\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha}$
$= (\lambda x_0.(\lambda x_1.(\lambda x'_0.(\lambda x'_1.x'_1) \cdot (x_1 \cdot x'_0)) \cdot (x_0 \cdot x_1)) \cdot (\lambda y_0.y_0)) \cdot (\lambda x.\mathbf{fix}\, x)$.

(3) $(\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}} = \lambda x_0.(\!|0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, []|\!)^{0:\mathsf{int}}_{\mathsf{int}} = \lambda x_0.\mathtt{Ifc}(x_0, (\lambda x'_0.x'_0) \cdot (\mathbf{succ}\, x_0))$.

(4) $(\!|\mathsf{Succ2}|\!)_{\mathsf{int}\to\mathsf{int}} = \big(\lambda x_0.(\lambda x_1.(\lambda x'_1.(\lambda x''_1.x''_1) \cdot (x_0 \cdot x'_1)) \cdot (x_0 \cdot x_1))\big) \cdot (\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}}$.

Note that, for all PCF programs P of type int, the program $(\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}} \cdot P$ is well typed and converges to a natural number exactly when P does.

Proposition 5.16. (1) *Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ and $\alpha \in \mathbb{T}$. If $\mathsf{M} : \alpha$ then $\vdash (\!|\mathsf{M}|\!)_\alpha : \alpha$.*

(2) *Let P be an EAM program, $T \in \mathcal{T}_{\mathbb{A}}$, $\Delta = i_1 : \alpha_{i_1}, \dots, i_k : \alpha_{i_k}$ be a type environment and $\alpha \in \mathbb{T}$. Then,*

$$\Delta \Vdash^r (P,T) : \alpha \;\Rightarrow\; x_{i_1} : \alpha_{i_1}, \dots, x_{i_k} : \alpha_{i_k} \vdash (\!|P,T|\!)^{i_1:\alpha_{i_1},\dots,i_k:\alpha_{i_k}}_\alpha : \alpha$$

Proof (Appendix A.3). Both items are proved by mutual induction on the height of a derivation of $\mathsf{M} : \alpha$ and of $\Delta \Vdash^r (P,T) : \alpha$. □

Theorem 5.17. *For all EAMs $\mathsf{M} : \alpha$, we have $|(\!|\mathsf{M}|\!)_\alpha| \equiv_\alpha \mathsf{M}$.*

Proof. One needs to consider the additional statement:

"For all $i_1 : \alpha_{i_1}, \dots, i_n : \alpha_{i_n} \Vdash^r (P,T) : \alpha$, $a_{i_1} \in \mathcal{D}_{\alpha_{i_1}}, \dots, a_{i_n} \in \mathcal{D}_{\alpha_{i_n}}$,

$$|(\!|P,T|\!)^{i_1:\alpha_{i_1},\dots,i_n:\alpha_{i_n}}_\alpha|_{x_{i_1},\dots,x_{i_n}} \mathbin{@} [a_{i_1}, \dots, a_{i_n}] \equiv_\alpha \langle \vec{R}^r_{a_{i_1},\dots,a_{i_n}}, P, T\rangle,$$

where $\vec{R}^r_{a_{i_1},\dots,a_{i_n}}$ denotes the list of registers $R_0, \dots, R_r$ such that, for all $j\,(0 \le j \le r)$, $!R_j = a_j$ if $j \in \{i_1, \dots, i_n\}$, and $!R_j = \varnothing$ otherwise."

The two statements are proven by mutual induction on a derivation of $\mathsf{M} : \alpha$ and $\Delta \Vdash^r (P, T) : \alpha$, respectively. (The details are worked out in the Technical Appendix A.3.) □

Corollary 5.18. *For all $a \in \mathcal{D}_\alpha$, there is a* PCF *program $\vdash P_a : \alpha$ such that $|P_a| \equiv_\alpha \#^{-1}(a)$.*

Theorem 5.19 (Completeness)**.** *Given two* PCF *programs P_1, P_2 of type α, we have*

$$P_1 \equiv_{\text{obs}} P_2 \Rightarrow |P_1| \equiv_\alpha |P_2|$$

Proof. Assume $P_1 \equiv_{\text{obs}} P_2$. Let $\alpha{=}\alpha_1 \to \cdots \to \alpha_n \to \mathsf{int}$ and $a_1, b_1 \in \mathcal{D}_{\alpha_1}, \ldots, a_n, b_n \in \mathcal{D}_{\alpha_n}$ be such that $\#^{-1}(a_i) \equiv_{\alpha_i} \#^{-1}(b_i)$. By Corollary 5.18, there are PCF programs $Q_1, \ldots, Q_n$, and $Q'_1, \ldots, Q'_n$ such that Q_i, Q'_i have type α_i and both $\#^{-1}(a_i) \equiv_{\alpha_i} |Q_i|$ and $\#^{-1}(b_i) \equiv_{\alpha_i} |Q'_i|$ hold, for every such i. We get

$$\begin{array}{rcll}
|P_1| \mathbin{@} [a_1, \ldots, a_n] & \equiv_{\mathsf{int}} & |P_1| \mathbin{@} [\#|Q_1|, \ldots, \#|Q_n|]\,, & \text{as } |P_1| \equiv_\alpha |P_1| \text{ by Lemma 5.8(1),} \\
& \equiv_{\mathsf{int}} & |P_1 \cdot Q_1 \cdots Q_n|, & \text{by Lemma 5.11,} \\
|P_1 \cdot Q_1 \cdots Q_n| \twoheadrightarrow_{\mathsf{c}} \mathsf{k} & \Leftrightarrow & P_1 \cdot Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{k}, & \text{by Theorem 4.10,} \\
& \Leftrightarrow & P_2 \cdot Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{k}, & \text{as } P_1 \equiv_{\text{obs}} P_2, \\
& \Leftrightarrow & |P_2 \cdot Q_1 \cdots Q_n| \twoheadrightarrow_{\mathsf{c}} \mathsf{k}, & \text{by Theorem 4.10,} \\
|P_2 \cdot Q_1 \cdots Q_n| & \equiv_{\mathsf{int}} & |P_2| \mathbin{@} [\#|Q_1|, \ldots, \#|Q_n|]\,, & \text{by Lemma 5.11,} \\
& \equiv_{\mathsf{int}} & |P_2| \mathbin{@} [b_1, \ldots, b_n]\,, & \text{as } |P_2| \equiv_\alpha |P_2| \text{ by Lemma 5.8(1).}
\end{array}$$

Conclude by transitivity (Lemma 5.8(1)). □

Adequacy and completeness together yield the main result of the paper: full abstraction.

Theorem 5.20 (Full Abstraction)**.** *The model $\mathscr{D}$ is fully abstract for* PCF*.*

Proof. By Definition 5.7, the interpretation in $\mathscr{D}$ is defined by $[\![P]\!]^\alpha = [\#|P|]_{\simeq_\alpha}$. Therefore, the full abstraction property follows directly from Theorems 5.10 and 5.19. □

Acknowledgment

The authors wish to acknowledge fruitful discussions with Flavien Breuvart and Damiano Mazza.

References

[AB17] Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, page 4–16, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3131851.3131855.

[ABM14] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. *SIGPLAN Not.*, 49(9):363–376, aug 2014. doi:10.1145/2692915.2628154.

[AC98] Roberto M. Amadio and Pierre-Louis Curien. *Domains and lambda-calculi*, volume 46 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1998.

[Acc18] Beniamino Accattoli. Proof nets and the linear substitution calculus. In Bernd Fischer and Tarmo Uustalu, editors, *Theoretical Aspects of Computing - ICTAC 2018 - 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, volume 11187 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.

[ACCL90] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 31–46. ACM Press, 1990. doi:10.1145/96709.96712.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991. doi:10.1017/S0956796800000186.

[AFFM07] Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. The power of closed reduction strategies. *Electron. Notes Theor. Comput. Sci.*, 174(10):57–74, 2007. doi:10.1016/j.entcs.2007.02.047.

[AK12] Beniamino Accattoli and Delia Kesner. The permutative λ-calculus. In Nikolaj S. Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2012. doi:10.1007/978-3-642-28717-6_5.

[ALV23] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. A log-sensitive encoding of Turing Machines in the λ-calculus. *CoRR*, abs/2301.12556, 2023. arXiv:2301.12556, doi:10.48550/ARXIV.2301.12556.

[AMJ94] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full abstraction for PCF. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1994. doi:10.1007/3-540-57887-0_87.

[BCL85] Gérard Berry, Pierre-Louis Curien, and Jean-Jacques Lévy. Full abstraction for sequential languages: state of the art. In M. Nivat and J. Reynolds, editors, *Algebraic methods in semantics*, pages 89–132. Cambridge University Press, 1985.

[Can13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Technical report, 2013.

[CHL96] Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM*, 43(2):362–397, 1996. doi:10.1145/226643.226675.

[Cre91] Pierre Cregut. *Machines a environnement pour la reduction symbolique et l'evaluation partielle*. PhD thesis, Université Paris-Diderot (Paris VII), 1991. In French. URL: http://www.theses.fr/1991PA077152.

[Cur92] Pierre-Louis Curien. Observable algorithms on concrete data structures. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 432–443. IEEE Computer Society, 1992. doi:10.1109/LICS.1992.185554.

[Cur07] Pierre-Louis Curien. Definability and full abstraction. *Electron. Notes Theor. Comput. Sci.*, 172:301–310, 2007. doi:10.1016/j.entcs.2007.02.011.

[Dav58] Martin Davis. *Computability & Unsolvability*. Dover Publications, New York, 1958.

[Del97] Giuseppe Della Penna. Una semantica operazionale per il network computing: le macchine di Turing virtuali. Master's thesis, Università degli Studi di L'Aquila, 1996-97. In Italian.

[DIM22] Giuseppe Della Penna, Benedetto Intrigila, and Giulio Manzonetto. Addressing machines as models of λ-calculus. *Log. Methods Comput. Sci.*, 18(3), 2022.

[FW87] Jon Fairbairn and Stuart Wray. Tim: A simple, lazy abstract machine to execute supercombinators. In Gilles Kahn, editor, *Functional Programming Languages and Computer Architecture*, pages 34–45, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. doi:10.1137/0218012.

[HO00] J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

[IMM23] Benedetto Intrigila, Giulio Manzonetto, and Nicolas Münnich. Extended addressing machines for PCF, with explicit substitutions. volume 1 - Proceedings of MFPS XXXVIII, 2023. URL: https://entics.episciences.org/10533.

[Kri07] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher Order Symbol. Comput.*, 20(3):199–207, sep 2007. doi:10.1007/s10990-007-9018-9.

[Lan64] Peter J. Landin. The Mechanical Evaluation of Expressions. *The Computer Journal*, 6(4):308–320, 01 1964. doi:10.1093/comjnl/6.4.308.

[Lan07] Frederic Lang. Explaining the lazy Krivine machine using explicit substitution and addresses. *Higher-Order and Symbolic Computation*, 2007. URL: https://hal.inria.fr/inria-00198756.

[Ler90] Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

[LM99] Jean-Jacques Lévy and Luc Maranget. Explicit substitutions and programming languages. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, volume 1738 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 1999. doi:10.1007/3-540-46691-6_14.

[Lon95] John Longley. *Realizability toposes and language semantics*. Ph.D. thesis, University of Edinburgh, 1995.

[Mel95] Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*, TLCA '95, page 328–334, Berlin, Heidelberg, 1995. Springer-Verlag.

[Mil77] Robin Milner. Fully abstract models of typed λ-calculi. *Theor. Comput. Sci.*, 4(1):1–22, 1977. doi:10.1016/0304-3975(77)90053-6.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[MRS99] Michael Marz, Alexander Rohr, and Thomas Streicher. Full abstraction and universality via realisability. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 174–182. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782612.

[Nic94] Hanno Nickau. Hereditarily sequential functionals. In Anil Nerode and Yuri V. Matiyasevich, editors, *Logical Foundations of Computer Science, Third International Symposium, LFCS'94, St. Petersburg, Russia, July 11-14, 1994, Proceedings*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994. doi:10.1007/3-540-58140-5_25.

[Ong95] C.-H. Luke Ong. Correspondence between operational and denotational semantics of PCF. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 269–356. Oxford University Press, 1995.

[Pit97] Andrew M. Pitts. A note on logical relations between semantics and syntax. *Log. J. IGPL*, 5(4):589–601, 1997. doi:10.1093/jigpal/5.4.589.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977. doi:10.1016/0304-3975(77)90044-5.

[SI96] Jill Seaman and S. Purushothaman Iyer. An operational semantics of sharing in lazy evaluation. *Sci. Comput. Program.*, 27(3):289–322, 1996. doi:10.1016/0167-6423(96)00012-3.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Vis80] Albert Visser. Numerations, λ-calculus, and arithmetic. In Hindley and Seldin, editors, *Essays on Combinatory Logic, Lambda-Calculus, and Formalism*, pages 259–284. Academic Press, San Diego, 1980.

Appendix A. Technical Appendix

This technical appendix is devoted to providing some omitted proofs. We often abbreviate 'induction hypothesis' as IH.

A.1. **PCF and omitted proofs from Section 2.** In order to prove Proposition 2.12, we need the following auxiliary lemma concerning w.h.-normalizing EPCF programs, that is, EPCF programs reducing to some value (by Remark 2.10(4)).

Lemma A.1. *Let $M, N, N_1, N_2 \in \mathcal{P}_{\mathsf{E}}$*

(1) *If $M \cdot N \twoheadrightarrow_{\mathtt{wh}} V$, then there exist M', σ such that $M \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M')^{\sigma}$ is a shorter reduction;*
(2) *If $\mathbf{pred}\, M \twoheadrightarrow_{\mathtt{wh}} V$ then there is a shorter reduction $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$;*
(3) *If $\mathbf{succ}\, M \twoheadrightarrow_{\mathtt{wh}} V$ then there is a shorter reduction $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$;*
(4) *If $\mathbf{ifz}(M, N_1, N_2) \twoheadrightarrow_{\mathtt{wh}} V$, then there is a shorter reduction $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$.*

Proof. (1) Since V cannot be an application, the reduction $M \cdot N \twoheadrightarrow_{\mathtt{wh}} V$ is non-empty. If $M \cdot N$ is a redex then M must have shape $(\lambda x.M')^{\sigma}$ for some M', σ and we are done. Otherwise, the reduction has shape $M \cdot N \to_{\mathtt{wh}} M' \cdot N \twoheadrightarrow_{\mathtt{wh}} V$ for some $M \to_{\mathtt{wh}} M'$. By repeating this reasoning we must obtain $M \twoheadrightarrow_{\mathtt{wh}} V'$, as the global reduction is finite. We conclude since $V' \cdot N$ must be a redex, and this is only possible if $V' = (\lambda x.M')^{\sigma}$.

(1-3) Analogous. □

Proof of Proposition 2.12. ($\Rightarrow$) By induction on the height of a derivation of $M \Downarrow^{\mathsf{E}} V$:

- $\underline{n} \Downarrow^{\mathsf{E}} \underline{n}$: Base case, by reflexivity $\underline{n} \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ holds.
- $(\lambda x.M)^{\sigma} \Downarrow^{\mathsf{E}} (\lambda x.M)^{\sigma}$: Base case, as above.
- $x^{\sigma} \Downarrow^{\mathsf{E}} V$ with $\sigma(x) = N$: Then $x^{\sigma} \to_{\mathtt{wh}} N$. Conclude by applying the IH to $N \Downarrow^{\mathsf{E}} V$.
- $(\mathbf{pred}\, M)^{\sigma} \Downarrow^{\mathsf{E}} \mathbf{0}$: We have $(\mathbf{pred}\, M)^{\sigma} \to_{\mathtt{wh}} \mathbf{pred}\, (M^{\sigma})$. By IH we obtain $M^{\sigma} \twoheadrightarrow_{\mathtt{wh}} \mathbf{0}$, from which it follows $\mathbf{pred}\, (M^{\sigma}) \twoheadrightarrow_{\mathtt{wh}} \mathbf{pred}\, \mathbf{0} \to_{\mathtt{wh}} \mathbf{0}$.
- $(\mathbf{pred}\, M)^{\sigma} \Downarrow^{\mathsf{E}} \underline{n}$: proceed as above, using $\mathbf{pred}\, (\underline{n+1}) \to_{\mathtt{wh}} \underline{n}$.
- $(\mathbf{succ}\, M)^{\sigma} \Downarrow^{\mathsf{E}} \underline{n+1}$: We have $(\mathbf{succ}\, M)^{\sigma} \to_{\mathtt{wh}} \mathbf{succ}\, (M^{\sigma})$. By IH we obtain $M^{\sigma} \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, so we conclude, remembering that $\underline{n} = \mathbf{succ}^{n}(\mathbf{0})$.
- $(\mathbf{ifz}(L, M, N))^{\sigma} \Downarrow^{\mathsf{E}} V_1$: Then, we have $(\mathbf{ifz}(L, M, N))^{\sigma} \to_{\mathtt{wh}} \mathbf{ifz}(L^{\sigma}, M^{\sigma}, N^{\sigma})$. By IH we get $L^{\sigma} \twoheadrightarrow_{\mathtt{wh}} \underline{0}$, so $\mathbf{ifz}(L^{\sigma}, M^{\sigma}, N^{\sigma}) \twoheadrightarrow_{\mathtt{wh}} M^{\sigma}$. Conclude since $M^{\sigma} \twoheadrightarrow_{\mathtt{wh}} V_1$ holds by IH.
- $(\mathbf{ifz}(L, M, N))^{\sigma} \Downarrow^{\mathsf{E}} V_2$: Proceed as above.
- $(\mathbf{fix}\, M)^{\sigma} \Downarrow^{\mathsf{E}} V$: We have $(\mathbf{fix}\, M)^{\sigma} \to_{\mathtt{wh}} \mathbf{fix}\, (M^{\sigma}) \to_{\mathtt{wh}} M^{\sigma} \cdot (\mathbf{fix}\, (M^{\sigma}))$. Conclude by IH.
- $(M \cdot N)^{\sigma} \Downarrow^{\mathsf{E}} V$: We have $(M \cdot N)^{\sigma} \to_{\mathtt{wh}} M^{\sigma} \cdot N^{\sigma}$. From the IH, we obtain $M^{\sigma} \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M')^{\sigma'}$. Since $(\lambda x.M')^{\sigma'} \cdot N^{\sigma} \to_{\mathtt{wh}} (M')^{\sigma'}\langle N^{\sigma}/x\rangle$, we may conclude using the IH.

($\Leftarrow$) We proceed by induction on the length $k = |M \twoheadrightarrow_{\mathtt{wh}} V|$.

Case $k = 0$. Then $M = V$. Apply either ($\mathrm{nat}^{\mathsf{E}}$) or ($\lambda^{\mathsf{E}}$), depending on the shape of V.

Case $k > 0$. Then $M \to_{\mathtt{wh}} N \twoheadrightarrow_{\mathtt{wh}} V$, for some $N \in \Lambda^{\mathsf{E}}$ with $|N \twoheadrightarrow_{\mathtt{wh}} V| = k - 1 < k$. By Remark 2.10(3), there exist a unique evaluation context $\mathsf{E}\Box$ and $M', N' \in \Lambda^{\mathsf{E}}$ such that $M = \mathsf{E}[M']$, $N = \mathsf{E}[N']$ and $M' \to N'$, where $\to$ represents the contraction of a computation or a percolation redex. Proceed by cases on the structure of $\mathsf{E}\Box$.

Subcase $\mathsf{E}\Box = \Box$ ($M = M', N = N'$). First, consider the computation reduction cases:

- $M = (\lambda x.M_1)^{\sigma} M_2$ and $N = M_1^{\sigma}\langle M_2/x\rangle$: By ($\lambda^{\mathsf{E}}$) we obtain $(\lambda x.M_1)^{\sigma} \Downarrow^{\mathsf{E}} (\lambda x.M_1)^{\sigma}$ and, from the IH, $M_1^{\sigma}\langle M_2/x\rangle \Downarrow^{\mathsf{E}} V$. We apply ($\beta_v^{\mathsf{E}}$) to infer that $(\lambda x.M_1)^{\sigma} M_2 \Downarrow^{\mathsf{E}} V$.
- $M = \mathbf{pred}\, \mathbf{0}$ and $N = \mathbf{0}$: By ($\mathrm{nat}^{\mathsf{E}}$) we obtain $\mathbf{0} \Downarrow^{\mathsf{E}} \mathbf{0}$. We apply ($\mathrm{pr}_0{}^{\mathsf{E}}$) to infer $\mathbf{pred}\, \mathbf{0} \Downarrow^{\mathsf{E}} \mathbf{0}$.

- The other computation cases for **pred** and **succ** are analogous to the above case.
- $M = \mathbf{ifz}(\mathbf{0}, L_1, L_2)$ and $N = L_1$: By $(\mathrm{nat}^{\mathsf{E}})$ we obtain $\mathbf{0} \Downarrow^{\mathsf{E}} \mathbf{0}$, and by the IH $L_1 \Downarrow^{\mathsf{E}} V$ holds. We apply $(\mathrm{ifz}_0{}^{\mathsf{E}})$ to infer that $\mathbf{ifz}(\mathbf{0}, L_1, L_2) \Downarrow^{\mathsf{E}} V$.
- $M = \mathbf{ifz}(\underline{n+1}, L_1, L_2)$ and $N = L_2$: Analogous to the above case.
- $M = \mathbf{fix}\, L$ and $N = L \cdot (\mathbf{fix}\, L)$: By IH we obtain $L \cdot (\mathbf{fix}\, L) \Downarrow^{\mathsf{E}} V$. We apply $(\mathrm{fix}^{\mathsf{E}})$ to infer $\mathbf{fix}\, L \Downarrow^{\mathsf{E}} V$.

Second, the percolation reduction steps:

- $M = x^{\sigma}$ and $\sigma(x) = N$: By IH we obtain $N \Downarrow^{\mathsf{E}} V$. We apply $(\mathrm{var}^{\mathsf{E}})$ to infer $x^{\sigma} \Downarrow^{\mathsf{E}} V$.
- $M = \mathbf{0}^{\sigma}$ and $N = \mathbf{0}$: We apply $(\mathrm{nat}^{\mathsf{E}})$ to infer $\mathbf{0}^{\sigma} \Downarrow^{\mathsf{E}} \mathbf{0}$.
- $M = (\mathbf{pred}\, M_1)^{\sigma}$ and $N = \mathbf{pred}\, (M_1^{\sigma})$: By Lemma A.1(2), $\mathbf{pred}\, (M_1^{\sigma}) \twoheadrightarrow_{\mathtt{wh}} V$ entails that there is a shorter reduction $M_1^{\sigma} \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $n \in \mathbb{N}$. By IH, we get $M_1^{\sigma} \Downarrow^{\mathsf{E}} \underline{n}$ and we conclude by applying $(\mathrm{pr}_0{}^{\mathsf{E}})$ or $(\mathrm{pr}^{\mathsf{E}})$ depending on whether $n = 0$ or not.
- Proceed as in the above case for all other percolation reduction steps.

Subcase $\mathsf{E}\Box = \mathbf{pred}\, (\mathsf{E}')$, for some context $\mathsf{E}'\Box$. As $M = \mathbf{pred}\, (\mathsf{E}'[M'])$ reduces to a value, by Lemma A.1(2) we must have a shorter reduction $\mathsf{E}'[M'] \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $n \in \mathbb{N}$. By IH we obtain $\mathsf{E}'[M'] \Downarrow^{\mathsf{E}} \underline{n}$, so we conclude by applying either $(\mathrm{pr}_0{}^{\mathsf{E}})$ or $(\mathrm{pr}^{\mathsf{E}})$.

Subcase $\mathsf{E}\Box = \mathbf{succ}\, (\mathsf{E}')$, for some $\mathsf{E}'\Box$. It follows analogously from IH and $(\mathrm{sc}^{\mathsf{E}})$.

Subcase $\mathsf{E}\Box = \mathsf{E}'L$, for some $\mathsf{E}'\Box$. Since $M \twoheadrightarrow_{\mathtt{wh}} V$, the reduction must factorize as

$$M = (\mathsf{E}'[M'])L \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M_1)^{\sigma} L \to_{\mathtt{wh}} M_1^{\sigma}[L/x] \twoheadrightarrow_{\mathtt{wh}} V$$

with both $\mathsf{E}'[M'] \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M_1)^{\sigma}$ and $M_1^{\sigma}[L/x] \twoheadrightarrow_{\mathtt{wh}} V$ shorter than k (see Lemma A.1(1)). By IH, we get $\mathsf{E}'[M'] \Downarrow^{\mathsf{E}} (\lambda x.M_1)^{\sigma}$ and $M_1^{\sigma}[L/x] \Downarrow^{\mathsf{E}} V$, so we conclude by (β_v^{E}).

Subcase $\mathsf{E}\Box = \mathbf{ifz}(\mathsf{E}', M_1, M_2)$, for some $\mathsf{E}'\Box$. Since $M \twoheadrightarrow_{\mathtt{wh}} V$, Lemma A.1(4) entails that the reduction must factorize as

$$M = \mathbf{ifz}(\mathsf{E}'[L], M_1, M_2) \twoheadrightarrow_{\mathtt{wh}} \mathbf{ifz}(\underline{n}, M_1, M_2) \to_{\mathtt{wh}} \begin{cases} M_1 \twoheadrightarrow_{\mathtt{wh}} V, & \text{if } n = 0, \\ M_2 \twoheadrightarrow_{\mathtt{wh}} V, & \text{otherwise,} \end{cases}$$

with $\mathsf{E}'[L] \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ and $|M_i \twoheadrightarrow_{\mathtt{wh}} V| < k$. By IH, we get either $[\mathsf{E}'[L] \Downarrow^{\mathsf{E}} \mathbf{0}$ and $M_1 \Downarrow^{\mathsf{E}} V]$ or $[\mathsf{E}'[L] \Downarrow^{\mathsf{E}} \underline{n}+1$ and $M_2 \Downarrow^{\mathsf{E}} V]$ (for some $n \in \mathbb{N}$). We conclude by either (ifz_0) or $(\mathrm{ifz}_{>0})$. □

Proof of Lemma 2.16. (1) (Syntax directedness) Trivial proof by inspection.

(2) (Strengthening) An easy proof by induction on the shape of M proves this in both directions - the statement holds for $\Gamma \vdash \mathbf{0} : \mathsf{int}$ and $\Gamma, x : \alpha \vdash x : \alpha$, and all other cases are derived from those two.

(3) (Subject reduction) We will prove this by induction on the shape of $M = \mathsf{E}[M']$ where M' is the contracted redex. The only interesting cases are when M' is in head position, i.e. $\mathsf{E}\Box = \Box$. For cases where σ is present, we will assume that $\mathrm{dom}(\sigma) = \{y_1, \ldots, y_n\}$.
   - Cases $M = x$ and $M = \mathbf{0}$ do not apply, as neither can reduce.
   - Case $M = (\lambda x.M_1)^{\sigma} \cdot M_2$ and $N = M_1^{\sigma}\langle M_2/x\rangle$. From $\vdash (\lambda x.M_1)^{\sigma} \cdot M_2 : \alpha$ we obtain the judgements $\vdash \sigma(y_1) : \gamma_1, \ldots, \vdash \sigma(y_n) : \gamma_n$, $y_1 : \gamma_1, \ldots, y_n : \gamma_n, x : \beta \vdash M_1 : \alpha$ and $\vdash M_2 : \beta$, via a single application of the $(\to_{\mathrm{E}})$ rule followed by repeated applications of the (σ) rule. Using all of these judgements we can then derive the judgement $\vdash M_1\langle M_2/x\rangle : \alpha$, through $n+1$ applications of the (σ) rule.
   - All other computation reduction cases are trivial, as they are identical to PCF where subject reduction holds.
   - Case $M = x^{\sigma}$, where $\sigma(x) = N$: Using (σ) we obtain the judgement $\vdash N : \alpha$.
   - Case $M = y^{\sigma}$, where $y \notin \mathrm{dom}(\sigma)$: Does not apply, as this term is not closed.

- Case $M = \mathbf{0}^\sigma$: Trivial, as this has the type int and $\mathbf{0}$ has the type int.
- Case $M = (\mathbf{pred}\, M_1)^\sigma$: We obtain the judgements $\vdash \sigma(y_1) : \gamma_1, \ldots, \vdash \sigma(y_n) : \gamma_n$, $y_1 : \gamma_1, \ldots, y_n : \gamma_n \vdash M_1 : \mathsf{int}$. We can reassemble these to form the judgement $\vdash M_1^\sigma : \mathsf{int}$, from which we derive $\vdash \mathbf{pred}\,(M_1^\sigma) : \mathsf{int}$.
- All other percolation reduction cases proceed identically to $(\mathbf{pred}\, M_1)^\sigma$.

The cases where $\mathsf{E}\Box \neq \Box$ are trivial applications of the induction hypothesis. □

Proof of Proposition 2.19. (1) By induction on a derivation of $M \to_{\mathsf{cr}} N$.

- Base cases.
  - Case $M = (\lambda x.L_1)^\sigma \cdot L_2$ and $N = L_1^\sigma\langle L_2/x\rangle$ where, say, $\sigma = \langle \vec{Y}/\vec{y}\rangle$ with $x \notin \vec{y}$. Then, we have:

$$\begin{array}{rcl} M^\dagger & = & (\lambda x.L_1^\dagger)[\vec{Y}^\dagger/\vec{y}] \cdot L_2^\dagger = (\lambda x.L_1^\dagger[\vec{Y}^\dagger/\vec{y}]) \cdot L_2^\dagger \quad \to_{\mathsf{PCF}} \quad L_1^\dagger[\vec{Y}^\dagger/\vec{y}][L_2^\dagger/x] \\ & = & (L_1^\sigma\langle L_2/x\rangle)^\dagger = N^\dagger \end{array}$$

  - Case $M = \mathbf{ifz}(\underline{0}, N, L)$. Then $L^\dagger = \mathbf{ifz}(\underline{0}, N^\dagger, L^\dagger) \to_{\mathsf{PCF}} N^\dagger$.
  - Case $M = \mathbf{ifz}(\underline{n+1}, L, N)$. Then $L^\dagger = \mathbf{ifz}(\underline{n+1}, L^\dagger, N^\dagger) \to_{\mathsf{PCF}} N^\dagger$.
  - All other base cases hold trivially.
- Case $M = M_1 \cdot M_2$ and $N = N_1 \cdot M_2$ with $M_1 \to_{\mathsf{cr}} N_1$. By induction hypothesis, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Therefore $M^\dagger = M_1^\dagger \cdot M_2^\dagger \to_{\mathsf{PCF}} N_1^\dagger \cdot M_2^\dagger = (N_1 \cdot M_2)^\dagger = N^\dagger$.
- Case $M = \mathbf{ifz}(M_1, L_1, L_2)$ and $N = \mathbf{ifz}(N_1, L_1, L_2)$ with $M_1 \to_{\mathtt{wh}} N_1$. By induction hypothesis, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Thus $M^\dagger = \mathbf{ifz}(M_1^\dagger, L_1^\dagger, L_2^\dagger) \to_{\mathsf{PCF}} \mathbf{ifz}(N_1^\dagger, L_1^\dagger, L_2^\dagger) = (\mathbf{ifz}(N_1, L_1, L_2))^\dagger = N^\dagger$.
- Case $M = \mathbf{pred}\, M_1$ and $N = \mathbf{pred}\, N_1$ with $M_1 \to_{\mathsf{cr}} N_1$. By induction hypothesis, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Thus $M^\dagger = \mathbf{pred}\,(M_1^\dagger) \to_{\mathsf{PCF}} \mathbf{pred}\,(N_1^\dagger) = (\mathbf{pred}\, N_1)^\dagger = N^\dagger$.
- Case $M = \mathbf{succ}\, M_1$ and $N = \mathbf{succ}\, N_1$ with $M_1 \to_{\mathsf{cr}} N_1$. Analogous.

(2) By induction on a derivation of $M \to_{\mathsf{pr}} N$.

- Base cases.
  - Case $M = (x\langle N/x\rangle)^\sigma$. Since $N \in \mathcal{P}_{\mathsf{E}}$, we have $\mathrm{FV}(N) = \emptyset$. Thus $M^\dagger = x[N^\dagger/x] = N^\dagger$.
  - Case $M = (y\langle L/x\rangle)^\sigma$ and $N = y^\sigma$. Since $M \in \mathcal{P}_{\mathsf{E}}$ we must have $y \in \mathrm{dom}(\sigma)$ with, say, $\sigma(y) = L' \in \mathcal{P}_{\mathsf{E}}$. Conclude since $M^\dagger = L'^\dagger = (y^\sigma)^\dagger = N^\dagger$.
  - All other base cases hold trivially.
- Case $M = M_1 \cdot M_2$ and $N = N_1 \cdot M_2$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Therefore $M^\dagger = M_1^\dagger \cdot M_2^\dagger = N_1^\dagger \cdot M_2^\dagger = (N_1 \cdot M_2)^\dagger = N^\dagger$.
- Case $M = \mathbf{ifz}(M_1, L_1, L_2)$ and $N = \mathbf{ifz}(N_1, L_1, L_2)$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Thus $M^\dagger = \mathbf{ifz}(M_1^\dagger, L_1^\dagger, L_2^\dagger) = \mathbf{ifz}(N_1^\dagger, L_1^\dagger, L_2^\dagger) = (\mathbf{ifz}(N_1, L_1, L_2))^\dagger = N^\dagger$.
- Case $M = \mathbf{pred}\, M_1$ and $N = \mathbf{pred}\, N_1$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Thus $M^\dagger = \mathbf{pred}\,(M_1^\dagger) = \mathbf{pred}\,(N_1^\dagger) = (\mathbf{pred}\, N_1)^\dagger = N^\dagger$.
- Case $M = \mathbf{succ}\, M_1$ and $N = \mathbf{succ}\, N_1$ with $M_1 \to_{\mathsf{pr}} N_1$. Analogous. □

A.2. **Omitted proofs from Section 4.**

Proof of Theorem 4.6. Proceed by induction on a derivation of $\Gamma \vdash M : \alpha$. We split into cases depending on the last applied rule.

- Case (0). In this case $M = \mathbf{0}$ and $\alpha = \mathsf{int}$. By Definition 4.4, we have $|\mathbf{0}|_{\vec{x}} = \mathsf{Pr}_1^{n+1} \mathbin{@} [0]$. By Lemma 4.3(1), we know that $\mathsf{Pr}_1^{n+1} : \mathsf{int} \to \vec{\delta} \to \mathsf{int}$. By rule (nat), we get $\mathsf{0} : \mathsf{int}$. By Proposition 3.17(1), we conclude $\mathsf{Pr}_1^{n+1} \mathbin{@} [0] : \vec{\delta} \to \mathsf{int}$;
- Case (ax). Then $M = x_i$ for some $x_i \in \vec{x}$ and $|x_i|_{\vec{x}} = \mathsf{Pr}_i^n$. By Lemma 4.3(1).
- Case ($\to_{\mathrm{E}}$). Then $M = M_1 \cdot M_2$ and there is $\beta \in \mathbb{T}$ such that $\Gamma \vdash M_1 : \beta \to \alpha, \Gamma \vdash M_2 : \beta$. From the induction hypothesis, we obtain that $|M_1|_{\vec{x}} : \vec{\delta} \to \beta \to \alpha$ and $|M_2|_{\vec{x}} : \vec{\delta} \to \beta$. By Lemma 4.3(1), we get $\mathsf{Pr}_1^1 : (\beta \to \alpha) \to \beta \to \alpha$. By Lemma 4.3(2), we know that $\mathsf{Apply}_n^2 : ((\beta \to \alpha) \to \beta \to \alpha) \to (\vec{\delta} \to \beta \to \alpha) \to \vec{\delta} \to \beta$. By Definition 4.4, we have $|M_1 \cdot M_2|_{\vec{x}} = \mathsf{Apply}_n^2 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}]$ so we conclude by Proposition 3.17(1).
- Case (σ). Then $M = M'\langle N/y\rangle$ with $\Gamma, y : \beta \vdash M' : \alpha$ and $\vdash N : \beta$, for some $\beta \in \mathbb{T}$. By induction hypothesis, we get $|M'|_{\vec{x},y} : \beta \to \vec{\delta} \to \alpha$ and $|N| : \beta$. By Definition 4.4, we have $|M'\langle N/y\rangle|_{\vec{x}} = |M'|_{y,\vec{x}} \mathbin{@} [\#|N|]$. Conclude by Proposition 3.17(1).
- Case ($\to_{\mathrm{I}}$). Then $M = \lambda y.N$ and $\alpha = \alpha_1 \to \alpha_2$, with $\Gamma, y : \alpha_1 \vdash N : \alpha_2$. By Definition 4.4, we have $|\lambda y.N|_{\vec{x}} = |N|_{\vec{x},y}$ so the case follows from the induction hypothesis.
- Case (+). Then $M = \mathbf{succ}\, N$ and $\alpha = \mathsf{int}$, with $\Gamma \vdash N : \mathsf{int}$. By induction hypothesis $|N|_{\vec{x}} : \vec{\delta} \to \mathsf{int}$ and, by Lemma 4.3(4), we have $\mathsf{Succ} : \mathsf{int} \to \mathsf{int}$. By Definition 4.4, $|\mathbf{succ}\, N|_{\vec{x}} = \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Succ}, \#|N|_{\vec{x}}]$. Conclude by Lemma 4.3(2) and Proposition 3.17(1).
- Case (−). Analogous to the previous case, using Lemma 4.3(3) instead of Lemma 4.3(2).
- Case (ifz). Then $M = \mathbf{ifz}(L, N_1, N_2)$ with $\Gamma \vdash L : \mathsf{int}$ and $\Gamma \vdash N_i : \alpha$, for each $i = 1, 2$. By induction hypothesis, we get $|L|_{\vec{x}} : \vec{\delta} \to \mathsf{int}$ and $|N_i|_{\vec{x}} : \vec{\delta} \to \alpha$, for every such i. By Lemma 4.3(5), we have $\mathsf{Ifz} : \mathsf{int} \to \alpha \to \alpha \to \alpha$. Since, by Definition 4.4, $|\mathbf{ifz}(L, N_1, N_2)|_{\vec{x}} = \mathsf{Apply}_n^3 \mathbin{@} [\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|N_1|_{\vec{x}}, \#|N_2|_{\vec{x}}]$ we conclude by applying Lemma 4.3(2) and Proposition 3.17(1).
- Case (Y). Then $M = \mathbf{fix}\, N$ with $\Gamma \vdash N : \alpha \to \alpha$. By induction hypothesis, we have $|N|_{\vec{x}} : \vec{\gamma} \to \alpha \to \alpha$. By rule (fix) we know that $\mathsf{Y} : (\alpha \to \alpha) \to \alpha$. The result follows from $|\mathbf{fix}\, N|_{\vec{x}} = \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Y}, \#|N|_{\vec{x}}]$ using Lemma 4.3(2) and Proposition 3.17(1). □

Proof of Proposition 4.9. By induction on a derivation of $M \to_{\mathtt{wh}} N$.

- Base cases. In the following $\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$ for some $n > 0$. As a matter of notation, we introduce the abbreviations $\vec{x} = x_1, \ldots, x_n$ and $\#\vec{\sigma} = \#|N_1|, \ldots, \#|N_n|$.
  - Case $M = ((\lambda y.M_1)^\sigma)M_2$ and $N = M_1^\sigma\langle M_2/x\rangle$. Wlog $y \notin \mathrm{dom}(\sigma)$ and since $M_1 \in \mathcal{P}_{\mathsf{E}}$, we must have $\mathrm{FV}(M_1) \subseteq \{y\}$. Therefore

$$\begin{array}{rcll} |M| & = & |((\lambda y.M_1)^\sigma)M_2| & \\ & = & \mathsf{Apply}_0^2 \mathbin{@} [\#|M_1^\sigma|_y, \#|M_2|], & \text{by Definition 4.4,} \\ & = & \mathsf{Pr}_1^1 \mathbin{@} [\#|M_1^\sigma|_y, \#|M_2|], & \text{since } \mathsf{Apply}_0^2 = \mathsf{Pr}_1^1 \text{ by Definition 4.1,} \\ & \to_{\mathsf{c}}^{\geq 2} & |M_1^\sigma|_y \mathbin{@} [\#|M_2|], & \text{by Lemma 4.2(1),} \\ & = & |M_1^\sigma\langle M_2/y\rangle|, & \text{by Definition 4.4.} \end{array}$$

  - Case $M = \mathbf{0}^\sigma$ and $N = \mathbf{0}$. On the one side, $|\mathbf{0}^\sigma| = |\mathbf{0}|_{\vec{x}} = \mathsf{Pr}_1^{n+1} \mathbin{@} [0, \#\vec{\sigma}]$ whence, by Lemma 4.2(1), we get $\mathsf{Pr}_1^{n+1} \mathbin{@} [0, \#\vec{\sigma}] \to_{\mathsf{c}}^{n+2} \#^{-1}(0) = \mathsf{0}$. On the other side, by Lemma 4.2(1), we obtain $|\mathbf{0}| = \mathsf{Pr}_1^1 \mathbin{@} [0] \to_{\mathsf{c}}^{\geq 2} \mathsf{0}$. Since $n > 0$, we conclude $|\mathbf{0}^\sigma| \succ_{\mathsf{c}} |\mathbf{0}|$.
  - Case $M = \mathbf{ifz}(\mathbf{0}, M_1, M_2)$ and $N = M_1$. Therefore $|M|$ is equal to

$$\begin{array}{rcll} |\mathbf{ifz}(\mathbf{0}, M_1, M_2)| & = & \mathsf{Apply}_0^3 \mathbin{@} [\#\mathsf{Ifz}, 0, \#|M_1|, \#|M_2|], & \text{by Definition 4.4,} \\ & = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Ifz}, 0, \#|M_1|, \#|M_2|], & \text{by Definition 4.1,} \\ & \to_{\mathsf{c}}^{\geq 2} & \mathsf{Ifz} \mathbin{@} [0, \#|M_1|, \#|M_2|], & \text{by Lemma 4.2(1),} \\ & \to_{\mathsf{c}} & |M_1|. & \end{array}$$

– Case $M = \mathbf{ifz}(\underline{k+1}, M_1, M_2)$, for some $k \geq 0$, and $N = M_2$. Therefore $|M|$ is equal to

$$\begin{array}{lll} & |\mathbf{ifz}(\underline{k+1}, M_1, M_2)| & \\ = & \mathsf{Apply}_0^3 \,@\, [\#\mathsf{Ifz}, k+1, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.4,} \\ = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Ifz}, k+1, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.1,} \\ \twoheadrightarrow_{\mathsf{c}}^2 & \mathsf{Ifz} \,@\, [k+1, \#|M_1|, \#|M_2|]\,, & \text{by Lemma 4.2(1),} \\ \twoheadrightarrow_{\mathsf{c}} & |M_2|. & \end{array}$$

– Case $M = y^\sigma$ for $y \notin \mathrm{dom}(\sigma)$. Vacuous, since $M \in \mathcal{P}_{\mathsf{E}}$.
– Case $M = x_i^\sigma$ and $N = \sigma(x_i) = N_i$. So $|x^\sigma| = \mathsf{Pr}_i^n \,@\, [\#\vec{\sigma}] \twoheadrightarrow_{\mathsf{c}}^{n+1} |N_i|$ by Lemma 4.2(1).
– Case $M = (M_1 \cdot M_2)^\sigma$ and $N = M_1^\sigma \cdot M_2^\sigma$. Since $M \in \mathcal{P}_{\mathsf{E}}$, each M_i^σ is closed, whence $\mathrm{FV}(M_i) \subseteq \{x_1, \ldots, x_n\}$. On the one side, we get

$$\begin{array}{lll} & |(M_1 \cdot M_2)^\sigma| = \mathsf{Apply}_n^2 @ [\#\mathsf{Pr}_1^1, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}, \#\vec{\sigma}] & \\ \twoheadrightarrow_{\mathsf{c}}^{10n+2} & \mathsf{Pr}_1^1 \,@\, [\#(|M_1|_{\vec{x}} @ [\#\vec{\sigma}]), \#(|M_2|_{\vec{x}} @ [\#\vec{\sigma}])]\,, & \text{by Lemma 4.2(2),} \\ \twoheadrightarrow_{\mathsf{c}}^2 & |M_1^\sigma| @ [\#|M_2^\sigma|], & \text{by Lemma 4.2(1).} \end{array}$$

On the other side, we get $|M_1^\sigma \cdot M_2^\sigma| = \mathsf{Pr}_1^1 @ [\#\mathsf{Pr}_1^1, \#|M_1^\sigma|, \#|M_2^\sigma|] \twoheadrightarrow_{\mathsf{c}}^4 |M_1^\sigma| @ [\#|M_2^\sigma|]$ by applying Lemma 4.2(1) (twice). Conclude since $10n + 4 > 4$ when $n > 0$.
– Case $M = \mathbf{pred}\,\mathbf{0}$ and $N = \mathbf{0}$. By Lemma 4.2(1) and (3), we have

$$\begin{array}{rl} |\mathbf{pred}\,\mathbf{0}| = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Pred}, 0] \twoheadrightarrow_{\mathsf{c}}^2 \mathsf{Pred} \,@\, [0] \\ \to_{\mathsf{c}} & \langle R_0 = 0, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [] \rangle \\ \to_{\mathsf{c}} & \langle R_0 = 0, \mathtt{Call}\ 0, [] \rangle \to_{\mathsf{c}} \mathsf{0} = |\mathbf{0}| \end{array}$$

– Case $M = \mathbf{pred}\,(\mathbf{succ}\,\underline{n})$ and $N = \underline{n}$. By applying Lemma 4.2(1) and (4), we obtain $|\mathbf{succ}\,\underline{n}| = \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Succ}, n] \twoheadrightarrow_{\mathsf{c}}^2 \mathsf{Succ} \,@\, [n] \to_{\mathsf{c}} \langle R_0 = n, 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [] \rangle \to_{\mathsf{c}} \langle R_0 = n+1, \mathtt{Call}\ 0, [] \rangle \to_{\mathsf{c}} \mathsf{n+1}$.
Using Lemma 4.2(3) as well, we conclude

$$\begin{array}{rl} |\mathbf{pred}\,(\mathbf{succ}\,\underline{n})| = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Pred}, \#|\mathbf{succ}\,\underline{n}|] \twoheadrightarrow_{\mathsf{c}}^2 \mathsf{Pred} \,@\, [\#|\mathbf{succ}\,\underline{n}|] \\ \to_{\mathsf{c}} & \langle R_0 = \#|\mathbf{succ}\,\underline{n}|, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [] \rangle \\ \twoheadrightarrow_{\mathsf{c}} & \langle R_0 = n+1, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [] \rangle \\ \twoheadrightarrow_{\mathsf{c}} & \langle R_0 = n, \mathtt{Call}\ 0, [] \rangle \to_{\mathsf{c}} \mathsf{n} = |\underline{n}| \end{array}$$

– Case $M = \mathbf{fix}\,M'$ and $N = M'(\mathbf{fix}\,M')$. In this case, we get

$$\begin{array}{lll} |\mathbf{fix}\,M'| = & \mathsf{Y} \,@\, [\#|M|]\,, & \text{by Definition 4.4,} \\ \twoheadrightarrow_{\mathsf{c}}^5 & |M'| \,@\, [\#\mathsf{Y} \,@\, [\#|M'|]]\,, & \text{by Lemma 4.2(6),} \\ = & |M'| \,@\, [\#|\mathbf{fix}\,M'|]\,, & \text{by Definition 4.4,} \\ \twoheadleftarrow_{\mathsf{c}}^2 & \mathsf{Pr}_1^1 \,@\, [\#|M'|, \#|\mathbf{fix}\,M'|]\,, & \text{by Lemma 4.2(1),} \\ \twoheadleftarrow_{\mathsf{c}}^2 & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Pr}_1^1, \#|M'|, \#|\mathbf{fix}\,M'|]\,, & \text{by Lemma 4.2(1),} \\ = & |M'(\mathbf{fix}\,M')|, & \text{by Definition 4.4.} \end{array}$$

– Case $M = (\mathbf{ifz}(L, M_1, M_2))^\sigma$ and $N = \mathbf{ifz}(L^\sigma, M_1^\sigma, M_2^\sigma)$. Note that $M \in \mathcal{P}_{\mathsf{E}}$ entails $L^\sigma, M_1^\sigma, M_2^\sigma$ closed, thus $\mathrm{FV}(L) \subseteq \{\vec{x}\}$ and $\mathrm{FV}(M_i) \subseteq \{\vec{x}\}$. By Lemma 4.2(2), we get

$$\begin{array}{lll} |(\mathbf{ifz}(L, M_1, M_2))^\sigma| & = & \mathsf{Apply}_n^3 @ [\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}, \#\vec{\sigma}] \\ & \twoheadrightarrow_{\mathsf{c}}^{13n+2} & \mathsf{Ifz} \,@\, [\#|L^\sigma|, \#|M_1^\sigma|, \#|M_2^\sigma|] \\ & \twoheadleftarrow_{\mathsf{c}}^2 & \mathsf{Pr}_1^1 @ [\#\mathsf{Ifz}, \#|L^\sigma|, \#|M_1^\sigma|, \#|M_2^\sigma|] \\ & = & |\mathbf{ifz}(L^\sigma, M_1^\sigma, M_1^\sigma)| \end{array}$$

Conclude since $13n + 2 > 2$ when $n > 0$.
– Case $M = (\mathbf{pred}\,M')^\sigma$ and $N = \mathbf{pred}\,((M')^\sigma)$. Analogous to the above.

– Case $M = (\mathbf{succ}\, M')^\sigma$ and $N = \mathbf{succ}\,((M')^\sigma)$. Analogous to the above.
– Case $M = (\mathbf{fix}\, M')^\sigma$ and $N = |\mathbf{fix}\,((M')^\sigma)|$. By Lemma 4.2(2), we obtain $|(\mathbf{fix}\, M')^\sigma| = \mathsf{Apply}_n^1 \mathbin{@} [\mathsf{Y}, \#|M'|_{\vec{x}}, \#\vec{\sigma}] \twoheadrightarrow_{\mathsf{c}}^{7n+2} \mathsf{Y} \mathbin{@} [\#|(M')^\sigma|] = |\mathbf{fix}\,(M'^\sigma)|$.

- Case $M = M_1 \cdot M_2$ and $N = M_1' \cdot M_2$, where $M_1 \to_{\mathtt{wh}} M_1'$. By IH $|M_1| \succ_{\mathsf{c}} |M_1'|$, i.e. there exists an EAM X such that $|M_1| \twoheadrightarrow_{\mathsf{c}}^k \mathsf{X}$ and $|M_1'| \twoheadrightarrow_{\mathsf{c}}^{k'} \mathsf{X}$, for some $k > k'$. Then we get

$$\begin{array}{rcll} |M_1 \cdot M_2| & = & \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1|, \#|M_2|], & \text{by Definitions 4.4 and 4.1,} \\ & \twoheadrightarrow_{\mathsf{c}}^4 & |M_1|@[\#|M_2|], & \text{by Lemma 4.2(2)} \\ & \twoheadrightarrow_{\mathsf{c}}^k & \mathsf{X}@[\#|M_2|] & \\ & \overset{k'}{\underset{\mathsf{c}}{\twoheadleftarrow}} & |M_1'|@[\#|M_2|] & \\ & \overset{4}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1'|, \#|M_2|], & \text{by Lemma 4.2(2),} \\ & = & |M_1 \cdot M_2|, & \text{by Definition 4.4.} \end{array}$$

- Case $M = \mathbf{ifz}(L, M_1, M_2)$ and $N = \mathbf{ifz}(L', M_1, M_2)$, where $L \to_{\mathtt{wh}} L'$. By IH $|L| \succ_{\mathsf{c}} |L'|$, i.e. there is an EAM X such that $|L| \twoheadrightarrow_{\mathsf{c}}^k \mathsf{X}$ and $|L'| \twoheadrightarrow_{\mathsf{c}}^{k'} \mathsf{X}$ for some $k > k'$. Then we get

$$\begin{array}{rcll} & |\mathbf{ifz}(L, M_1, M_2)| & & \\ = & \mathsf{Pr}_1^1@[\#\mathsf{Ifz}, \#|L|, \#|M_1|, \#|M_2|], & & \text{by Def. 4.4 and 4.1,} \\ \twoheadrightarrow_{\mathsf{c}}^2 & \mathsf{Ifz}@[\#|L|, \#|M_1|, \#|M_2|], & & \text{by Lemma 4.2(2)} \\ \twoheadrightarrow_{\mathsf{c}}^3 & \left\langle \begin{array}{r} R_0 = \#|L|, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\,1,\,2); \mathtt{Call}\ 0, [] \end{array} \right\rangle, & & \text{by Lemma 4.2(5)} \\ \twoheadrightarrow_{\mathsf{c}}^k & \left\langle \begin{array}{r} R_0 = \#\mathsf{X}, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\,1,\,2); \mathtt{Call}\ 0, [] \end{array} \right\rangle, & & \\ \overset{k'}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \left\langle \begin{array}{r} R_0 = \#|L'|, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\,1,\,2); \mathtt{Call}\ 0, [] \end{array} \right\rangle & & \\ \overset{3}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Ifz}@[\#|L|, \#|M_1|, \#|M_2|], & & \text{by Lemma 4.2(5),} \\ \overset{2}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1'|, \#|M_2|], & & \text{by Lemma 4.2(2),} \\ = & |M_1' \cdot M_2|, & & \text{by Definition 4.4.} \end{array}$$

- Case $M = \mathbf{pred}\, M'$ and $N = \mathbf{pred}\, N'$, where $M' \to_{\mathtt{wh}} N'$. Analogous.
- Case $M = \mathbf{succ}\, M'$ and $N = \mathbf{succ}\, N'$, where $M' \to_{\mathtt{wh}} N'$. Analogous. □

A.3. **Omitted proofs from Section 5.**

Proof of Lemma 5.12. By induction on a derivation of $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \beta$. As a matter of notation, we let $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$ and introduce the abbreviations

$$\begin{array}{rclcrcl} \vec{a} & = & a_1, \ldots, a_n; & & \vec{x} & = & x_1, \ldots, x_n; \\ \vec{a}^- & = & a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n; & & \vec{x}^- & = & x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n. \end{array}$$

- Case $\Gamma \vdash \underline{0} : \mathsf{int}$. We get

$$\begin{array}{rcll} |\underline{0}|_{\vec{x}}@[\vec{a}] & = & \mathsf{Pr}_1^{n+1} \mathbin{@} [0, \vec{a}]\,, & \text{by Definition 4.4,} \\ & \to_{\mathsf{c}} & \#^{-1}(0), & \text{by Lemma 4.2(1),} \\ & \overset{}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Pr}_1^n \mathbin{@} [0, \vec{a}^-]\,, & \text{by Lemma 4.2(1),} \\ & = & |\underline{0}|_{\vec{x}^-}@[\vec{a}^-], & \text{by Definition 4.4.} \end{array}$$

Conclude by Lemma 5.8(2).

- Case $\Gamma, y : \beta \vdash y : \beta$. Let $b \in \mathcal{D}_\beta$. By Definition 4.4 and Lemma 4.2(1), we have $|y|_{\vec{x},y} @ [\vec{a}, b] = \mathsf{Pr}^{n+1}_{n+1} @ [\vec{a}, b] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(b)$ and $|y|_{\vec{x}^-,y} @ [\vec{a}^-, b] = \mathsf{Pr}^n_n @ [\vec{a}^-, b] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(b)$. Conclude by Lemma 5.8(2).
- Case $\Gamma \vdash M \cdot N : \beta$ since $\Gamma \vdash M : \alpha \to \beta$ and $\Gamma \vdash N : \alpha$. Then, we have

$$\begin{array}{rcll} |M \cdot N|_{\vec{x}} @ [\vec{a}] & = & \mathsf{Apply}^2_n @ [\#\mathsf{Pr}^1_1, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}, \vec{a}], & \text{by Definition 4.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & |M|_{\vec{x}} @ [\vec{a}, \#(|N|_{\vec{x}} @ [\vec{a}])], & \text{by Lemma 4.2(2).} \end{array}$$

  By IH, we have $|N|_{\vec{x}} @ [\vec{a}] \equiv_\alpha |N|_{\vec{x}^-} @ [\vec{a}^-]$ and $|M|_{\vec{x}} @ [\vec{a}] \equiv_{\alpha\to\beta} |M|_{\vec{x}^-} @ [\vec{a}^-]$, so by Lemma 5.8(1) (reflexivity), we derive $|M|_{\vec{x}} @ [\vec{a}, \#(|N|_{\vec{x}} @ [\vec{a}])] \equiv_\beta |M|_{\vec{x}} @ [\vec{a}, \#(|N|_{\vec{x}^-} @ [\vec{a}^-])]$. Conclude by Lemmas 5.8(1)-(2) and Definition 4.4.
- Case $\Gamma \vdash M\langle N/z\rangle : \beta$ with $\Gamma, z : \alpha \vdash M : \beta$ and $\vdash N : \alpha$. By Definition 4.4 we get $|M\langle N/z\rangle|_{\vec{x}} @ [\vec{a}] = |M|_{z,\vec{x}} @ [\#|N|, \vec{a}]$. Conclude by applying the IH.
- Case $\Gamma \vdash \lambda z.M : \gamma \to \delta$ since $\Gamma, z : \gamma \vdash M : \delta$. By Definition 4.4 we get $|\lambda z.M|_{\vec{x}} = |M|_{\vec{x},z}$. This case follows straightforwardly from the IH.
- Case $\Gamma \vdash \mathbf{succ}\, M : \mathsf{int}$, since $\Gamma \vdash M : \mathsf{int}$. We get

$$\begin{array}{rcll} |\mathbf{succ}\, M|_{\vec{x}} @ [\vec{a}] & = & \mathsf{Apply}^1_n @ [\#\mathsf{Succ}, \#|M|_{\vec{x}}, \vec{a}], & \text{by Definition 4.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \mathsf{Succ} @ [\#(|M|_{\vec{x}} @ [\vec{a}])], & \text{by Lemma 4.2(2).} \end{array}$$

  From the IH we obtain $|M|_{\vec{x}} @ [\vec{a}] \equiv_{\mathsf{int}} |M|_{\vec{x}^-} @ [\vec{a}^-]$, so by Lemma 5.8(1) (reflexivity),

$$\mathsf{Succ} @ [\#(|M|_{\vec{x}} @ [\vec{a}])] \equiv_{\mathsf{int}} \mathsf{Succ} @ [\#(|M|_{\vec{x}^-} @ [\vec{a}^-])]$$

  Conclude by Lemmas 5.8-(1)(2) and Definition 4.4.
- Case $\Gamma \vdash \mathbf{pred}\, M : \mathsf{int}$. Analogous.
- Case $\Gamma \vdash \mathbf{ifz}(L, M, N) : \beta$. Analogous.
- Case $\Gamma \vdash \mathbf{fix}\, M : \beta$ since $\Gamma \vdash M : \beta \to \beta$. Then,

$$\begin{array}{rcll} |\mathbf{fix}\, M|_{\vec{x}} @ [\vec{a}] & = & \mathsf{Apply}^1_n @ [\#\mathsf{Y}, \#|M|_{\vec{x}}, \vec{a}], & \text{by Definition 4.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \mathsf{Y} @ [\#(|M|_{\vec{x}} @ [\vec{a}])], & \text{by Lemma 4.2(2).} \end{array}$$

  By IH, we have $|M|_{\vec{x}} @ [\vec{a}] \equiv_{\beta\to\beta} |M|_{\vec{x}^-} @ [\vec{a}^-]$, so by Lemma 5.8(1)(reflexivity) we get

$$\mathsf{Y} @ [\#(|M|_{\vec{x}} @ [\vec{a}])] \equiv_\beta \mathsf{Y} @ [\#(|M|_{\vec{x}^-} @ [\vec{a}^-])]$$

  Conclude by Definition 4.4, applying Lemma 5.8(1)(2) in case $n > 1$ and Lemma 5.8(1) when $n = 1$. □

Proof of Proposition 5.16. Both (1) and (2) follow by mutual induction on a derivation of $\mathsf{M} : \alpha$ and $\Delta \Vdash^r (P, T) : \alpha$ and call IH1, IH2 the respective induction hypothesis.

As a matter of notation, if $\Delta = i_1 : \alpha_{i_1}, \ldots, i_k : \alpha_{i_k}$ we let $\Delta^* = x_{i_1} : \alpha_{i_1}, \ldots, x_{i_k} : \alpha_{i_k}$.

(1) Case (nat). Then $\mathsf{M} = \mathsf{n}$ and $\alpha = \mathsf{int}$. Conclude since $(\!|\mathsf{n}|\!)_{\mathsf{int}} = \underline{n}$ and $\vdash \underline{n} : \mathsf{int}$ holds.

Case (fix). Then $\mathsf{M} = \mathsf{Y}$ and $\alpha = (\beta \to \beta) \to \beta$. We type $(\!|\mathsf{Y}|\!)^\Delta_\alpha = \lambda x.\mathbf{fix}\, x$ as follows:

$$\dfrac{\dfrac{\dfrac{}{x : \beta \to \beta \vdash x : \beta \to \beta}\,(\mathrm{ax})}{x : \beta \to \beta \vdash \mathbf{fix}\, x : \beta}\,(\mathrm{Y})}{\vdash \lambda x.\mathbf{fix}\, x : (\beta \to \beta) \to \beta}\,(\to_{\mathrm{I}})$$

Case $(\vec{R})$. Then $\mathsf{M} = \langle R_0, \ldots, R_r, P, T\rangle$ with $\vec{R} \models \Delta$ and $\Delta \Vdash^r (P, T) : \alpha$, for some $\Delta = i_1 : \alpha_{i_1}, \ldots, i_k : \alpha_{i_k}$. From the former condition, by rules $(R_\varnothing)$ and $(R_\mathbb{T})$, we get a derivation of $\#^{-1}(!R_{i_j}) : \alpha_{i_j}$ having smaller size, for all $1 \le j \le k$. By applying IH1,

we obtain a derivation of $\vdash (\!|\#^{-1}(!R_{i_j})|\!)_{\alpha_{i_j}} : \alpha_{i_j}$. From the latter condition and IH2, we have $\Delta^* \vdash (\!|P,T|\!)^{\Delta}_{\alpha} : \alpha$. Therefore, we construct a derivation

$$\dfrac{\dfrac{\Delta^* \vdash (\!|P,T|\!)^{\Delta}_{\alpha} : \alpha}{\vdash \lambda x_{i_1} \dots x_{i_k}.(\!|P,T|\!)^{\Delta}_{\alpha} : \alpha_{i_1} \to \dots \to \alpha_{i_k} \to \alpha} \quad \vdash (\!|\#^{-1}(!R_{i_j})|\!)_{\alpha_{i_j}} : \alpha_{i_j} \quad 1 \le j \le k}{\vdash (\lambda x_{i_1} \dots x_{i_k}.(\!|P,T|\!)^{\Delta}_{\alpha}) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_k})|\!)_{\beta_{i_k}} : \alpha}$$

(2) Case ($\text{load}_{\varnothing}$). Then $P = \texttt{Load}\ j; P'$, $T = []$, $\alpha = \beta_1 \to \beta_2$ and $\Delta[j : \beta_1] \Vdash^r (P', []) : \beta_2$ has a derivation of smaller size. There are two subcases.

- Case $j \notin \text{dom}(\Delta)$, whence $\Delta[j : \beta_1] = \Delta, j : \beta_1$. By IH2 we get $\Delta^* \vdash (\!|P', []|\!)^{\Delta, j:\beta_1}_{\beta_2} : \beta_2$. Simply apply rule ($\to_{\text{I}}$).
- Case $j \in \text{dom}(\Delta)$, say, $j = i_k$. From the IH2 we obtain $\Gamma, x_{i_k} : \beta_1 \vdash (\!|P', []|\!)^{\Delta[i_k:\beta_1]}_{\beta_2} : \beta_2$ for $\Gamma = x_{i_1} : \alpha_{i_1}, \dots, x_{i_{k-1}} : \alpha_{i_{k-1}}$. By applying the rule ($\to_{\text{I}}$), we obtain a derivation of $\Gamma \vdash \lambda x_{i_k}.(\!|P', []|\!)^{\Delta[i_k:\beta_1]}_{\beta_2} : \beta_1 \to \beta_2$ whence, by strengthening (Lemma 2.16(2)), we conclude $\Gamma, x_{i_k} : \alpha_{i_k} \vdash \lambda x_{i_k}.(\!|P', []|\!)^{\Delta[i_k:\beta_1]}_{\beta_2} : \beta_1 \to \beta_2$.

Case ($\text{load}_{\mathbb{T}}$). Then $P = \texttt{Load}\ j; P'$, $T = a :: T'$. Moreover, $\Delta[j : \beta] \Vdash^r (P', T') : \alpha$ and $\#^{-1}(a) : \beta$ have a derivation of smaller size, for some β. From the former, one obtains a derivation of $\Delta^* \vdash \lambda x_j.(\!|P', T'|\!)^{\Delta[j:\beta]}_{\alpha} : \beta \to \alpha$ proceeding as above. By the IH1 applied to the latter, we obtain a derivation of $\vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$, whence $\Delta^* \vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$ holds by strengthening (Lemma 2.16(2)). By applying rule ($\to_{\text{E}}$), we conclude that $\Delta^* \vdash \left(\lambda x_j.(\!|P', T'|\!)^{\Delta[j:\beta]}_{\alpha}\right) \cdot (\!|\#^{-1}(a)|\!)_{\beta} : \alpha$ is derivable.

Case (pred). Then $P = j \leftarrow \texttt{Pred}(i); P'$ and $(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P', T') : \alpha$ has a smaller derivation. We assume $j \notin \text{dom}(\Delta, i : \mathsf{int})$, otherwise proceed as in case ($\text{load}_{\varnothing}$). By IH2, we obtain a derivation of $\Delta^*, x_i : \mathsf{int}, x_j : \mathsf{int} \vdash (\!|(P', T')|\!)^{\Delta, i:\mathsf{int}, j:\mathsf{int}}_{\alpha} : \alpha$, thus:

$$\dfrac{\dfrac{\dfrac{\Delta^*, x_i : \mathsf{int}, x_j : \mathsf{int} \vdash (\!|P', T'|\!)^{\Delta, i:\mathsf{int}, j:\mathsf{int}}_{\alpha} : \alpha}{\Delta^*, x_i : \mathsf{int} \vdash \lambda x_j.(\!|P', T'|\!)^{\Delta, i:\mathsf{int}, j:\mathsf{int}}_{\alpha} : \alpha} \quad \dfrac{\Delta^*, x_i : \mathsf{int} \vdash x_i : \mathsf{int}}{\Delta^*, x_i : \mathsf{int} \vdash \mathbf{pred}\, x_i : \mathsf{int}}}{\Delta^*, x_i : \mathsf{int} \vdash x_i : \mathsf{int} \qquad \Delta^*, x_i \vdash (\lambda x_j.(\!|P', T'|\!)^{\Delta, i:\mathsf{int}, j:\mathsf{int}}_{\alpha}) \cdot (\mathbf{pred}\, x_i) : \alpha}}{\Delta^*, x_i : \mathsf{int} \vdash \texttt{Ifc}(x_i, (\lambda x_j.(\!|P', T'|\!)^{\Delta, i:\mathsf{int}, j:\mathsf{int}}_{\alpha}) \cdot (\mathbf{pred}\, x_i)) : \alpha}$$

Case (succ). Analogous.

Case (call). Then $P = \texttt{Call}\ i$ and $T = [a_1, \dots, a_n]$ with $\#^{-1}(a_j) : \beta_j$, for j $(1 \le j \le n)$. Call $\Gamma = \Delta^*, x_i : \beta_1 \to \dots \to \beta_n \to \alpha$. By IH1, we get $\vdash (\!|\#^{-1}(a_j)|\!)_{\beta_j} : \beta_j$ whence $\Gamma \vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$ holds by strengthening (Lemma 2.16(2)). Derive

$$\dfrac{\Gamma \vdash x_i : \beta_1 \to \dots \to \beta_n \to \alpha \quad \Gamma \vdash (\!|\#^{-1}(a_1)|\!)_{\beta_1} : \beta_1 \cdots \Gamma \vdash (\!|\#^{-1}(a_n)|\!)_{\beta_n} : \beta_n}{\Gamma \vdash x_i \cdot (\!|\#^{-1}(a_1)|\!)_{\beta_1} \cdots (\!|\#^{-1}(a_n)|\!)_{\beta_n} : \alpha}$$

(3) Case (app). Then $P = l \leftarrow \texttt{App}(i, j); P'$ and $(\Delta, i : \beta_1 \to \beta_2, j : \beta_1)[l : \beta_2] \Vdash^r (P', T) : \alpha$ has a smaller derivation, for some β_1, β_2. Assume that $l \notin \text{dom}(\Delta, i : \beta_1 \to \beta_2, j : \beta_1)$, otherwise proceed as in case ($\text{load}_{\varnothing}$). Setting $\Gamma' = \Delta^*, x_i : \beta_1 \to \beta_2, x_j : \beta_1$, we get:

$$\dfrac{\dfrac{\Gamma, x_l : \beta_2 \vdash (\!|P', T|\!)^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2}_{\alpha} : \alpha}{\Gamma \vdash \lambda x_l.(\!|P', T|\!)^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2}_{\alpha} : \beta_2 \to \alpha} \quad \dfrac{\Gamma \vdash x_i : \beta_1 \to \beta_2 \quad \Gamma \vdash x_j : \beta_1}{\Gamma \vdash x_i \cdot x_j : \beta_2}}{\Gamma \vdash (\lambda x_l.(\!|P', T|\!)^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2}_{\alpha}) \cdot (x_i \cdot x_j) : \alpha}$$

Case (test). Then $P = m \leftarrow \mathtt{Test}(i,\, j,\, l); P'$ and $(\Delta, i : \mathsf{int}, j : \beta, l : \beta)[m : \beta] \Vdash^r (P, T) : \alpha$ has a smaller derivation, for some β. Assume $m \notin \mathrm{dom}(\Delta, i : \mathsf{int}, j : \beta, l : \beta)$, otherwise proceed as in case ($\mathrm{load}_\varnothing$). Let $\Gamma = \Delta^*, x_i : \mathsf{int}, x_j : \beta, x_l : \beta, x_m : \beta$, we get:

$$\dfrac{\Gamma \vdash x_i : \mathsf{int} \qquad \dfrac{\dfrac{\Gamma, x_m : \beta \vdash (\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta} : \alpha}{\Gamma \vdash \lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta} : \beta \to \alpha} \quad \dfrac{\Gamma \vdash x_n : \mathsf{int}, n \in \{i, j, k\}}{\Gamma \vdash \mathbf{ifz}(x_i, x_j, x_l) : \beta}}{\Gamma \vdash (\lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta}) \cdot \mathbf{ifz}(x_i, x_j, x_l) : \alpha}}{\Gamma \vdash \mathtt{Ifc}(x_i, (\lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta}) \cdot \mathbf{ifz}(x_i, x_j, x_l)) : \alpha}$$

This concludes the proof. □

Proof of Theorem 5.17. Consider the following statements:

(1) Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$. If $\mathsf{M} : \alpha$ then $|(\!|\mathsf{M}|\!)_\alpha| \equiv_\alpha \mathsf{M}$.

(2) For all $i_1 : \alpha_{i_1}, \ldots, i_n : \alpha_{i_n} \Vdash^r (P, T) : \alpha$, $a_{i_1} \in \mathcal{D}_{\alpha_{i_1}}, \ldots, a_{i_n} \in \mathcal{D}_{\alpha_{i_n}}$,

$$|(\!|P, T|\!)_\alpha^{i_1:\alpha_{i_1}, \ldots, i_n:\alpha_{i_n}}|_{x_{i_1}, \ldots, x_{i_n}} @ [a_{i_1}, \ldots, a_{i_n}] \equiv_\alpha \langle \vec{R}^r_{a_{i_1}, \ldots, a_{i_n}}, P, T\rangle,$$

where $\vec{R}^r_{a_{i_1}, \ldots, a_{i_n}}$ denotes the list of registers $R_0, \ldots, R_r$ such that, for all j ($0 \le j \le r$), $!R_j = a_j$ if $j \in \{i_1, \ldots, i_n\}$, and $!R_j = \varnothing$ otherwise.

Both statements are proven using simultaneous induction on a derivation of $\mathsf{M} : \alpha$ and $\Delta \Vdash^r (P, T) : \alpha$. We refer to the former induction hypothesis as IH1 and to the latter as IH2. As a matter of notation, we let $\Delta = i_1 : \beta_{i_1}, \ldots, i_n : \beta_{i_n}$, $\vec{x} = x_{i_1}, \ldots, x_{i_n}$, and $\vec{a} = a_{i_1}, \ldots, a_{i_n}$ such that for all $j \in \{i_1, \ldots, i_n\}$, $a_j \in \mathcal{D}_{\beta_j}$.

- Case $\mathsf{k} : \mathsf{int}$. We prove this case by induction on $k \in \mathbb{N}$ (and call this IH1′).
  - Case $k = 0$: By Definition 4.4 we get $|(\!|\mathsf{0}|\!)_{\mathsf{int}}| = |\underline{0}| = \mathsf{Pr}^1_1 @ [0]$. Conclude by Lemmas 4.2(1) and 5.8(2).
  - Case $k = m + 1$, for some $m \in \mathbb{N}$. Then by Definitions 4.4 and 5.14, we have
$$\begin{array}{rcl} |(\!|\mathsf{k}|\!)_{\mathsf{int}}| & = & |\underline{m+1}| = |\mathbf{succ}\, \underline{m}| \\ & = & \mathsf{Apply}^1_0 @ [\#\mathsf{Succ}, \#|\underline{m}|] \;= \mathsf{Pr}^1_1 @ [\#\mathsf{Succ}, \#|\underline{m}|]. \end{array}$$
    By IH1′ we have $|\underline{m}| \equiv_{\mathsf{int}} \mathsf{m}$, so $\mathsf{Pr}^1_1 @ [\#\mathsf{Succ}, \#|\underline{m}|] \equiv_{\mathsf{int}} \mathsf{Pr}^1_1 @ [\#\mathsf{Succ}, m]$, by reflexivity. Conclude by Lemma 4.2(1),(4), Lemma 5.8(2), and transitivity.
- Case $\mathsf{Y} : (\alpha \to \alpha) \to \alpha$. Let $a, b \in \mathcal{D}_{\alpha\to\alpha}$ such that $a \simeq_{\alpha\to\alpha} b$. Since by Lemma 4.3(1) and Lemma 5.8(2) we have $\mathsf{Pr}^1_1 @ [a] \equiv_{\alpha\to\alpha} \#^{-1}(a)$, we obtain
$$\begin{array}{rcll} |(\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha}| @ [a] & = & |\lambda x.\mathbf{fix}\, x| @ [a]\,, & \text{by Definition 5.14,} \\ & = & \mathsf{Apply}^1_1 @ [\#\mathsf{Y}, \#\mathsf{Pr}^1_1, a]\,, & \text{by Definition 4.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \#^{-1}(a) @ [\#\mathsf{Y} \cdot (\#\mathsf{Pr}^1_1 \cdot a)]\,, & \text{by Lemma 4.2(6),} \\ & \equiv_\alpha & \#^{-1}(a) @ [\#\mathsf{Y} \cdot a]\,, & \text{by Lemma 5.8(3),} \\ & \equiv_\alpha & \#^{-1}(a) @ [\#\mathsf{Y} \cdot b]\,, & \text{by reflexivity,} \\ & \equiv_\alpha & \#^{-1}(b) @ [\#\mathsf{Y} \cdot b]\,, & \text{by Definition 5.6,} \\ & {}_{\mathsf{c}}\!\twoheadleftarrow & \mathsf{Y} @ [b]\,, & \text{by Lemma 4.2(6).} \end{array}$$
  Conclude by Lemma 5.8(2) and transitivity.
- Case $\langle \vec{R}, P, T\rangle : \alpha$. Let $\vec{R} \models \Delta$. By Definition 5.14, Lemma 4.2, and Definition 4.4 we get
$$\begin{array}{rcl} |(\!|\langle \vec{R}, P, T\rangle|\!)_\alpha| & = & |(\lambda x_{i_1} \ldots x_{i_n}.(\!|P, T|\!)^\Delta_\alpha) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}| \\ & = & |(\!|P, T|\!)^\Delta_\alpha|_{\vec{x}} @ [\#|(\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}}|, \ldots, \#|(\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}|] \end{array}$$

By IH1, for all $k \in \mathrm{dom}(\Delta)$, we have $\#|(\!|\#^{-1}(!R_k)|\!)_{\beta_k}| \simeq_\alpha !R_k$. Then by reflexivity,

$$\begin{array}{l} |(\!|P,T|\!)^\Delta_\alpha|_{\vec{x}} \mathbin{@} [\#|(\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}}|, \ldots, \#|(\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}|] \\ \equiv_\alpha |(\!|P,T|\!)^\Delta_\alpha|_{\vec{x}} \mathbin{@} [!R_{i_1}, \ldots, !R_{i_n}] \end{array}$$

Conclude by IH2, Lemma 5.8(2), and transitivity.

- Case $\Delta \Vdash^r (\mathtt{Load}\ k; P, []) : \beta \to \alpha$. There are two subcases.
  - Subcase $k \notin \mathrm{dom}(\Delta)$. Let $b, c \in \mathcal{D}_\beta$ such that $b \simeq_\beta c$. We get

$$\begin{array}{lll} & |(\!|\mathtt{Load}\ k; P, []|\!)^\Delta_{\beta\to\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}, b] & \\ = & |\lambda x_k.(\!|P, []|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x}} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 5.14,} \\ = & |(\!|P, []|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 4.4,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a},b}, P, []\rangle, & \text{by IH2,} \\ {}_{\mathsf{c}}\!\leftarrow & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [b]\rangle, & \text{by Definition 3.10,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [c]\rangle, & \text{by reflexivity.} \end{array}$$

    Conclude by Lemma 5.8(2) and transitivity.
  - Subcase $k \in \mathrm{dom}(\Delta)$. Let $k = i_m$, and let $b, c \in \mathcal{D}_\beta$ such that $b \simeq_\beta c$. We also fix $\vec{x'} = x_{i_1}, \ldots, x_{i_{m-1}}, x_{i_{m+1}}, \ldots, x_{i_n}$ and $\vec{a'} = a_{i_1}, \ldots, a_{i_{m-1}}, a_{i_{m+1}}, \ldots, a_{i_n}$. We get

$$\begin{array}{lll} & |(\!|\mathtt{Load}\ k; P, []|\!)^\Delta_{\beta\to\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}, b] & \\ = & |\lambda x_k.(\!|P, []|\!)^{\Delta[k:\beta]}_\alpha|_{\vec{x}} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 5.14,} \\ \equiv_\alpha & |\lambda x_k.(\!|P, []|\!)^{\Delta[k:\beta]}_\alpha|_{\vec{x'}} \mathbin{@} [\vec{a'}, b]\,, & \text{by Lemma 5.12,} \\ = & |(\!|P, []|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x'},x_k} \mathbin{@} [\vec{a'}, b]\,, & \text{by Definition 4.4,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a'},b}, P, []\rangle, & \text{by IH2,} \\ {}_{\mathsf{c}}\!\twoheadleftarrow & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [b]\rangle, & \text{by Definition 3.10,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [c]\rangle, & \text{by reflexivity.} \end{array}$$

    Conclude by Lemma 5.8(2) and transitivity.

In the cases following, we assume that $k \notin \mathrm{dom}(\Delta)$. If $k \in \mathrm{dom}(\Delta)$, one proceeds as above.

- Case $\Delta \Vdash^r (\mathtt{Load}\ k; P, b :: T) : \alpha$. By Definition 5.14, Lemma 4.2, and Definition 4.4, we get

$$\begin{array}{lll} & |(\!|\mathtt{Load}\ k; P, b :: T|\!)^\Delta_\alpha|_{\vec{x}} \mathbin{@} [\vec{a}] & \\ = & |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\beta}_\alpha) \cdot (\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}] & \\ = & \mathsf{Apply}^2_n \mathbin{@} [\#\mathsf{Pr}^1_1, \#|(\!|P,T|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x},x_k}, \#|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}}, \vec{a}] & \\ \twoheadrightarrow_{\mathsf{c}} & |(\!|P,T|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,. & \end{array}$$

  By Proposition 5.16, $\vdash (\!|\#^{-1}(b)|\!)_\beta : \beta$, so by Corollary 5.13 and IH1, $\#|(\!|\#^{-1}(b)|\!)_\beta| \mathbin{@} \vec{a} \simeq_\alpha b$. By reflexivity, we obtain

$$|(\!|P,T|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] \equiv_\alpha |(\!|P,T|\!)^{\Delta,k:\beta}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, b]\,.$$

  Conclude by IH2, Lemma 5.8(2), and transitivity.

- Case $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \mathtt{Pred}(i_m); P, T) : \alpha$. Fix the notation $\mathsf{M} = |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha) \cdot (\mathbf{pred}\, x_{i_m})|_{\vec{x}}$. By Definition 5.14 and Lemma 4.2 we get

$$\begin{array}{ll} & |(\!|k \leftarrow \mathtt{Pred}(i_m); P, T|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] \\ = & |\mathtt{Ifc}(x_{i_m}, (\lambda x_k.(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha) \cdot (\mathbf{pred}\, x_{i_m}))|_{\vec{x}} \mathbin{@} [\vec{a}] \\ = & \mathsf{Apply}^3_n \mathbin{@} [\#\mathsf{Ifz}, \#\mathsf{Pr}^n_{i_m}, \#\mathsf{M}, \#\mathsf{M}, \vec{a}] \\ \twoheadrightarrow_{\mathsf{c}} & \mathsf{Ifz} \mathbin{@} [\#(\mathsf{Pr}^n_{i_m} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}] \\ \equiv_\alpha & \mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}]\,. \end{array}$$

  There are two subcases from this point.
  - Subcase $\#^{-1}(a_{i_m})$ does not terminate. Then, by Definition 3.10, we have that the machine $\mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}]$ and $\langle \vec{R}^r_{\vec{a}}, l \leftarrow \mathtt{Pred}(k); P, T\rangle$ cannot terminate either. By Definition 3.13, we have $\langle \vec{R}^r_{\vec{a}}, l \leftarrow \mathtt{Pred}(k); P, T\rangle : \alpha$. Conclude by Lemma 5.8(2) and transitivity.
  - Subcase $\#^{-1}(a_{i_m}) \twoheadrightarrow_{\mathsf{c}} \mathsf{t} = \#^{-1}(t)$, for some $t \in \mathbb{N}$. Now, easy calculations give $|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}] \twoheadrightarrow_{\mathsf{c}} \mathsf{t}' := \#^{-1}(t \ominus 1)$, from which $|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\mathsf{int}} \mathsf{t}'$ follows by Lemma 5.8(2). Then we get

$$\begin{array}{lll} & \mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}] & \\ \twoheadrightarrow_{\mathsf{c}} & |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha) \cdot (\mathbf{pred}\, x_{i_m})|_{\vec{x}} \mathbin{@} [\vec{a}]\,, & \text{by Lem. 4.2(6),} \\ = & \mathsf{Apply}^2_n \mathbin{@} \left[\begin{array}{l} \#\mathsf{Pr}^1_1, \#|(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha|_{\vec{x},x_k}, \\ \#|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}}, \vec{a} \end{array}\right], & \text{by Def. 4.4,} \\ \twoheadrightarrow_{\mathsf{c}} & |(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,, & \text{by Lem. 4.2(2),} \\ \equiv_\alpha & |(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_\alpha|_{\vec{x},x_k} \mathbin{@} [\vec{a}, t \ominus 1, 0)]\,, & \text{by reflexivity,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}[R_k := t \ominus 1], P, T\rangle, & \text{by IH2,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{Pred}(i_m); P, T\rangle, & \text{by Lem. 5.8(2).} \end{array}$$

  Conclude by Lemma 5.8(2) and transitivity.
- Case $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \mathtt{Succ}(i_m); P, T) : \alpha$. Analogous.
- Case $\Delta \Vdash^r (k \leftarrow \mathtt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T) : \alpha$, where $\Delta(l) = \mathsf{int}$, $\Delta(i_{m_1}) = \beta$, $\Delta(i_{m_2}) = \beta$. There are two subcases.
  - Subcase $\#^{-1}(a_{i_l})$ does not terminate. Proceed as in case $x_{i_1}\colon \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \mathtt{Pred}(i_m); P, T) : \alpha$.
  - Subcase $\#^{-1}(a_{i_l}) \twoheadrightarrow_{\mathsf{c}} \#^{-1}(t), t \in \mathbb{N}$. Proceed as in case $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \mathtt{Pred}(i_m); P, T) : \alpha$ to get
    $|(\!|k \leftarrow \mathtt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_\alpha$
    $|(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|\vec{x}, x_k \mathbin{@} [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]$. There are two subcases.
    * Subcase $t = 0$. By Lemma 5.8(2) we have $|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_\beta \#^{-1}(a_{i_{m_1}})$. Then we get

$$\begin{array}{lll} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] & \\ \equiv_\alpha & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, a_{i_{m_1}}]\,, & \text{by reflexivity,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_{m_1}}], P, T\rangle, & \text{by IH2,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T\rangle, & \text{by Lemma 5.8(2).} \end{array}$$

    Conclude by Lemma 5.8(2) and transitivity.

∗ Subcase $t > 0$. By Lemma 5.8(2) we have $|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} @ [\vec{a}] \equiv_\beta \#^{-1}(a_{i_{m_2}})$. Then we get

$$\begin{array}{lll} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} @ [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} @ [\vec{a}]\,)] & \\ \equiv_\alpha & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} @ [\vec{a}, a_{i_{m_2}}]\,, & \text{by reflexivity,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_{m_2}}], P, T\rangle, & \text{by IH2,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T\rangle, & \text{by Lemma 5.8(2).} \end{array}$$

Conclude by Lemma 5.8(2) and transitivity.

- Case $\Delta \Vdash^r (k \leftarrow \mathtt{App}(i_l,\, i_m); P, T) : \gamma$, where $\Delta(i_l) = \beta \to \alpha$ and $\Delta(i_m) = \beta$. By Definition 5.14, Definition 4.4, and Lemma 4.2(2), we get

$$\begin{array}{lll} & |(\!|m \leftarrow \mathtt{App}(i_l,\, i_m); P, T|\!)^{\Delta}_{\gamma}|_{\vec{x}} @ [\vec{a}] & \\ = & |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}) \cdot (x_{i_l} \cdot x_{i_m})|_{\vec{x}} @ [\vec{a}] & \\ = & \mathsf{Apply}^2_n @ [\#\mathsf{Pr}^1_1, \#|(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k}, \#|x_{i_l} \cdot x_{i_m}|_{\vec{x}}, \vec{a}] & \\ \twoheadrightarrow_{\mathsf{c}} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} @ [\vec{a}, \#(|x_{i_l} \cdot x_{i_m}|_{\vec{x}} @ [\vec{a}]\,)] & \end{array}$$

By Lemma 5.8(2) we have $\#\mathsf{Pr}^n_{i_m} @ [\vec{a}] \equiv_\beta \#^{-1}(a_{i_m})$. We then have

$$\begin{array}{lll} |x_{i_l} \cdot x_{i_m}|_{\vec{x}} @ [\vec{a}] \quad = & \mathsf{Apply}^2_n @ [\#\mathsf{Pr}^1_1, \#\mathsf{Pr}^n_{i_l}, \#\mathsf{Pr}^n_{i_m}, \vec{a}]\,, & \text{by Definition 4.4,} \\ \twoheadrightarrow_{\mathsf{c}} & \mathsf{Pr}^n_{i_l} @ [\vec{a}, \#(\#\mathsf{Pr}^n_{i_m} @ [\vec{a}]\,)]\,, & \text{by Lemma 4.2(2),} \\ \equiv_\alpha & \mathsf{Pr}^n_{i_l} @ [\vec{a}, a_{i_m}]\,, & \text{by reflexivity,} \\ \equiv_\alpha & \#^{-1}(a_{i_l}) @ [a_{i_m}]\,, & \text{by Lemma 5.8(2).} \end{array}$$

Thus we get

$$\begin{array}{lll} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} @ [\vec{a}, \#(|x_{i_l} \cdot x_{i_m}|_{\vec{x}} @ [\vec{a}]\,)] & \\ \equiv_\gamma & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} @ [\vec{a}, a_{i_l} \cdot a_{i_m}]\,, & \text{by reflexivity,} \\ \equiv_\gamma & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_l} \cdot a_{i_m}], P, T\rangle, & \text{by IH2,} \\ \equiv_\gamma & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{App}(i_l,\, i_m); P, T\rangle, & \text{by Lemma 5.8(2).} \end{array}$$

Conclude by Lemma 5.8(2) and transitivity.

- Case $\Delta \Vdash^r (\mathtt{Call}\ k, [b_1, \ldots, b_m]) : \alpha$, where $\Delta(k) = \gamma_1 \to \cdots \to \gamma_m \to \alpha$ and for all $0 < j \leq m$, $b_j \in \mathcal{D}_{\gamma_j}$. Let $\vec{b} = b_1, \ldots, b_m$. By Definition 5.14 we get $|(\!|\mathtt{Call}\ k, [\vec{b}]|\!)^{\Delta}_{\alpha}|_{\vec{x}} @ [\vec{a}] = |x_k \cdot (\!|\#^{-1}(b_1)|\!)_{\gamma_1} \cdots (\!|\#^{-1}(b_m)|\!)_{\gamma_m}|_{\vec{x}} @ [\vec{a}]$. By an easy induction on m, one shows the following:

$$|x_k \cdot (\!|\#^{-1}(b_1)|\!)_{\gamma_1} \cdots (\!|\#^{-1}(b_m)|\!)_{\gamma_m}|_{\vec{x}} @ [\vec{a}] \equiv_\alpha |x_k|_{\vec{x}} @ [\vec{a}, \vec{b}] \text{ (cf. Lemma 5.11).}$$

By Definition 4.4 and Lemma 4.2(1), we get $|x_k|_{\vec{x}} @ [\vec{a}, \vec{b}] = \mathsf{Pr}^n_k @ [\vec{a}, \vec{b}] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(a_k) @ [\vec{b}]$. Conclude by Lemma 5.8(2) and transitivity, as $\langle \vec{R}^r_{\vec{a}}, \mathtt{Call}\ k, [\vec{b}]\rangle \twoheadrightarrow_{\mathsf{c}} \#^{-1}(a_k) @ [\vec{b}]$. □