Logical Methods in Computer Science
Volume 18, Issue 3, 2022, pp. 10:1–10:31
https://lmcs.episciences.org/

Submitted Jul. 02, 2021
Published Jul. 29, 2022

ADDRESSING MACHINES AS MODELS OF λ-CALCULUS

GIUSEPPE DELLA PENNA [a], BENEDETTO INTRIGILA [b], AND GIULIO MANZONETTO [c]

[a] Dep. of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
e-mail address: giuseppe.dellapenna@univaq.it

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

[c] Univ. Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, CNRS, F-93430 Villetaneuse, France.
e-mail address: manzonetto@univ-paris13.fr

Abstract. Turing machines and register machines have been used for decades in theoretical computer science as abstract models of computation. Also the λ-calculus has played a central role in this domain as it allows to focus on the notion of functional computation, based on the substitution mechanism, while abstracting away from implementation details. The present article starts from the observation that the equivalence between these formalisms is based on the Church-Turing Thesis rather than an actual encoding of λ-terms into Turing (or register) machines. The reason is that these machines are not well-suited for modelling λ-calculus programs.

We study a class of abstract machines that we call *addressing machine* since they are only able to manipulate memory addresses of other machines. The operations performed by these machines are very elementary: load an address in a register, apply a machine to another one via their addresses, and call the address of another machine. We endow addressing machines with an operational semantics based on leftmost reduction and study their behaviour. The set of addresses of these machines can be easily turned into a combinatory algebra. In order to obtain a model of the full untyped λ-calculus, we need to introduce a rule that bares similarities with the ω-rule and the rule ζ_β from combinatory logic.

Introduction

In theoretical computer science several models of computation have been considered over the years, since the pioneering work of Turing [Tur36]. Turing Machines (TMs) certainly played a crucial role in the understanding of the notion of computation, while Register Machines (RMs) are more adapted to represent programs executed in a von Neumann architecture [Rog87]. From a recursion-theoretic perspective, the class of partial recursive functions provides a natural description of those numeric functions that can be calculated by a mechanical device [Kle36]. In mathematical logic, λ-calculus [Bar84] and the related formalism — combinatory logic [CF58] — proved to be an inexhaustible source of inspiration for the development of formal systems, proof assistants and functional programming languages. As it is well-known, the basic computational mechanism of λ-calculus is the *symbolic substitution*

Key words and phrases: Addressing machines, λ-calculus, combinatory algebras, λ-models.

of an expression for a variable. All these formalisms — and many others that have been subsequently introduced — are quite different, but they can be proved equivalent in the sense that they are capable of representing the same class of partial numerical functions, i.e. the class of partial recursive functions. Despite the enormous importance of this result — in particular as a strong evidence for the so called Turing-Church Thesis — it is still of great interest to understand, at a deeper level, the relationships between the different computational formalisms.

In particular, the relationship between λ-calculus and partial recursive functions was investigated by Henk Barendregt, who tried to build during his PhD a model of untyped λ-calculus (λ-model [Koy82, Mey82]) out of Kleene's partial combinatory algebra having the set of "codes" $\mathbb{N}$ as underlying set and as application the partial operator $\{x\}(y)$, that can be interpreted as the possible result of applying the partial computable function with code x to the input y. His intention was to use this binary operator $\{\cdot\}(\cdot)$ to construct a (total) combinatory algebra in such a way that Kleene's translation of λ-calculus results would become a simple model-theoretic interpretation. It is important to observe that a direct approach cannot work, as recursive functions implicitly use the classic computational model, which requires that a function is *strict* on its arguments, that is, the function is undefined whenever any of its arguments is undefined. On the other hand, both λ-calculus and combinatory logic allow the representation of non-strict functions such as the combinator $\mathbf{K}$.

Barendregt has set up several sophisticated constructions, but a definite solution is still missing. The problem is nowadays receiving the attention of the scientific community because of the recent republication of his PhD thesis [Bar71], extended with commentaries. On the bright side, these investigations led to the formulation of the famous ω-rule because — if such a λ-model exists — then it needs to satisfy this strong extensionality axiom.

Following the same line of research, but attacking the problem from a different angle, one might meaningfully wonder whether it is possible to construct a λ-model based on appropriate abstract machines. The most obvious and canonical choice would be considering Turing Machines, but such an attempt has the same problem as the one encountered with recursive function, since TMs are strict on their arguments. A second problem is how to represent higher-order computations: in an imperative programming language a function can take another function as argument by working with its address, but in a TM this would require to encode processes as data and then manipulate and execute such codes indirectly. This makes the simple, intuitive notion of communication through addresses extremely difficult to realize. To this day, no λ-model of this kind has ever been constructed.

In this article we define a class of abstract machines, where the notions of address and communication (through addresses) are not only crucial to model computation, but they become the unique ingredients available. These machines are called *addressing machines* and possess a finite tape from which they can read the input, some internal registers where they can store values read from the tape, and an internal program which is composed by a list of instructions that are executed sequentially. The input-tape and the internal registers are reminiscent of those in TMs and RMs, respectively. Every machine is uniquely identified by its address, which is a value taken from a fixed countable set $\mathbb{A}$. In this formalism, addresses are the only available data-type — this means that both the input-tape and the internal registers of a machine (once initialized) contain addresses from $\mathbb{A}$. Programs are written in an assembly language possessing only three instructions[1]. Besides reading its inputs, an

[1]This choice is made on purpose, in the attempt of determining the minimum amount of operations giving rise to a Turing-complete formalism.

addressing machine can apply two addresses a, b with each other and store the resulting address $a \cdot b$ in an internal register. Intuitively, $a \cdot b$ is obtained by first taking the machine M having address a, then appending b to its input-tape, and finally calculating the address of this new machine. This application operation being static and manipulating addresses exclusively is *total* even when the referenced machines are non-terminating once executed. As a last step of its execution, an addressing machine can transfer the computation to another machine, possibly extending its input-tape, by retrieving its address from a register. Although not crucial in the abstract definition of an addressing machine, it should be clear at this point that any implementation of this formalism requires the association between the machines and their addresses to be effective (see Section 6 for more details).

Addressing machines share with λ-calculus the fact that there is no fundamental distinction between processes and data-types: in order to perform calculations on natural numbers a machine needs to manipulate the addresses of the corresponding numerals. Another similarity is the fact that in both settings communication is achieved by transferring the computation from one entity to another one. In the case of addressing machines, the machine currently "in execution" transfers the control by calling the address of another machine. In λ-calculus, the subterm "in charge" is the one occupying the so-called "head position" and the control of the computation is transferred when the head variable is substituted by another term. It is worth noting that process calculi such as the π-calculus also address communication using the concept of channel, where messages are exchanged [Mil99, SW01]. This is not the kind of communication that we are going to model here: our form of communication is encoded in the notion of address, so that a machine receiving a message results in a new machine with a different address. In other words, we do not model the dynamics of the communication, but the evolution of the machine addresses actually encodes the effects of communication. Another difference is the fact that π-calculus naturally models parallel computations as well as concurrency, while addressing machines are designed for representing sequential computations (one machine at a time is executed).

Contents. The aim of the paper is twofold. On the one side we want to present the class of addressing machines and analyze their fundamental properties. This is done in Section 2, where we describe their operational semantics in two different styles: as a term rewriting system (small-step semantics) and as a set of inference rules (big-step semantics). The two approaches are shown to be equivalent in case of addressing machines executing a terminating program (Proposition 2.16). On the other side, we wish to construct a model of the untyped λ-calculus based on addressing machines, and study the interpretations of λ-terms. For this reason, we recall in the preliminary Section 1 the main facts about λ-calculus, its equational theories and denotational models. It turns out that the set $\mathbb{A}$ of addresses, together with the operation of application previously described, is not a combinatory algebra (nor, *a fortiori*, a λ-model). In Section 3 we show that it can be turned into a combinatory algebra by quotienting under an equivalence relation arising naturally from our small-step operational semantics. Two addresses are equivalent if the corresponding machines are interconvertible using a more liberal rewriting relation. From the confluence property enjoyed by this relation, we infer the consistency of the algebra (Proposition 3.12). Unfortunately, the combinatory algebra so-obtained is not yet a model of λ-calculus — there are still β-convertible λ-terms having different interpretations. Section 5 is devoted to showing that a λ-model actually arises when adding to the system a mild form of extensionality sharing similarities both with the ω-rule in λ-calculus [Bar71] and with the rule ζ_β from combinatory logic [HS86]. The consistency of the model follows from an analysis of the underlying ordinal. Interestingly, the model itself is not extensional (Theorem 4.10).

Related works. A preliminary version of addressing machines appeared in Della Penna's MSc thesis [Del97]. Other abstract machines having similar primitive instructions are present in the literature, but they were studied from the perspective of functional programs implementation, see e.g. [FW87]. We do not claim that addressing machines are innovative, the originality of our work relies on the construction of a λ-model (Section 5) and its analysis. The practice of associating an address to a term is also well-established in the implementation of functional programming languages, and can be seen as the practical counterpart of explicit substitutions [LM99, BLM05, ACGC19]. The relationship between our addressing machines and explicit substitutions will be discussed in Section 6.

1. Preliminaries

We present some notions that will be useful in the rest of the article.

1.1. **The Lambda Calculus — Its Syntax.**

For the λ-calculus we mainly follow Barendregt's first book [Bar84]. We consider fixed a countable set Var of *variables* denoted by $x, y, z, \dots$

Definition 1.1. The set Λ of λ-*terms* over Var is generated by the following simplified[2] grammar (for $x \in \mathrm{Var}$):

$$M, N, P, Q \ ::= \ x \mid \lambda x.M \mid MN \tag{Λ}$$

We assume that application is left-associative and has a higher precedence than λ-abstraction. Therefore $\lambda x.\lambda y.\lambda z.xyz$ stands for $(\lambda x.(\lambda y.(\lambda z.(xy)z)))$. Moreover, we often write $\lambda x_1 \dots x_n.M$ for $\lambda x_1 \dots \lambda x_n.M$.

Definition 1.2. Let $M \in \Lambda$.

(i) The set $\mathrm{FV}(M)$ of *free variables* of M is defined by induction:

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}, \\ \mathrm{FV}(\lambda x.P) & = & \mathrm{FV}(P) - \{x\}, \\ \mathrm{FV}(PQ) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q). \end{array}$$

(ii) We say that M is *closed*, or *a combinator*, whenever $\mathrm{FV}(M) = \emptyset$.

(iii) We let $\Lambda^o = \{M \in \Lambda \mid \mathrm{FV}(M) = \emptyset\}$ be the set of all combinators.

The variables occurring in M that are not free are called "bound". From now on, λ-terms are considered modulo α-*conversion*, namely, up to the renaming of bound variables (see [Bar84, §2.1]).

[2] This basically means that parentheses are left implicit.

Notation 1.3. Concerning specific combinators we let:

$$
\begin{array}{llll}
\mathbf{I} & = & \lambda x.x, & \text{identity,} \\
\mathbf{1} & = & \lambda xy.xy, & \text{an } \eta\text{-expansion of the identity,} \\
\mathbf{K} & = & \lambda xy.x, & \text{first projection,} \\
\mathbf{F} & = & \lambda xy.y, & \text{second projection,} \\
\mathbf{S} & = & \lambda xyz.xz(yz), & S\text{-combinator from Combinatory Logic,} \\
\mathbf{\Delta} & = & \lambda x.xx, & \text{self-application,} \\
\mathbf{\Omega} & = & \mathbf{\Delta\Delta}, & \text{paradigmatic looping combinator,} \\
\mathbf{Y} & = & \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)), & \text{Curry's fixed point combinator.}
\end{array}
$$

The λ-calculus is given by the set Λ endowed with reduction relations that turn it into a higher-order term rewriting system.

We say that a relation $\mathrm{R} \subseteq \Lambda^2$ is *compatible* if it is compatible w.r.t. application and λ-abstraction. This means that, for $M, N, P \in \Lambda$, if $M \,\mathrm{R}\, N$ holds then also $MP \,\mathrm{R}\, NP$, $PM \,\mathrm{R}\, PN$ and $\lambda x.M \,\mathrm{R}\, \lambda x.N$ hold.

Definition 1.4. Define the following reduction relations.

(i) The *β-reduction* $\to_\beta$ is the least compatible relation closed under the rule

$$(\lambda x.M)N \to M[N/x] \tag{β}$$

where $M[N/x]$ denotes the λ-term obtained by substituting N for all free occurrences of x in M, subject to the usual proviso about renaming bound variables in M to avoid capture of free variables in N.

(ii) Similarly, the *η-reduction* $\to_\eta$ is the least compatible relation closed under the rule

$$\lambda x.Mx \to M, \text{ if } x \notin \mathrm{FV}(M). \tag{η}$$

(iii) Moreover, we define $\to_{\beta\eta} = \to_\beta \cup \to_\eta$.

(iv) The relations $\to_\beta, \to_\eta$ and $\to_{\beta\eta}$ respectively generate the notions of *multi-step reduction* $\twoheadrightarrow_\beta, \twoheadrightarrow_\eta, \twoheadrightarrow_{\beta\eta}$ (resp. *conversion* $=_\beta, =_\eta, =_{\beta\eta}$) by taking the reflexive and transitive (and symmetric) closure.

Theorem 1.5 (Church-Rosser)**.** *The reduction relation $\twoheadrightarrow_{\beta(\eta)}$ is confluent:*

$$M \twoheadrightarrow_{\beta(\eta)} M_1 \;\wedge\; M \twoheadrightarrow_{\beta(\eta)} M_2 \quad \Rightarrow \quad \exists N \in \Lambda\,.\, M_1 \twoheadrightarrow_{\beta(\eta)} N \;{}_{\beta(\eta)}\!\twoheadleftarrow M_2$$

The λ-terms are classified into solvable and unsolvable, depending on their capability of interaction with the environment.

Definition 1.6. A λ-term M is called *solvable* if $(\lambda\vec{x}.M)\vec{P} =_\beta \mathbf{I}$ for some $\vec{x}$ and $\vec{P} \in \Lambda$. Otherwise M is called *unsolvable*.

We say that a λ-term M *has a head normal form* (*hnf*) if it reduces to a λ-term of shape $\lambda x_1 \dots x_n.yM_1 \cdots M_k$ for some $n, k \geq 0$. As shown by Wadsworth in [Wad76], a λ-term M is solvable if and only if M has a head normal form. The typical examples of unsolvable λ-terms are $\mathbf{\Omega}, \lambda x.\mathbf{\Omega}$ and $\mathbf{YI}$.

1.2. **Lambda theories and lambda models.** Conservative extensions of β-conversion are known as "λ-theories" and have been extensively studied in the literature, see e.g. [Bar84, LS04, IMP19, IS17, MPSS19].

Definition 1.7.

(i) A *λ-theory* $\mathcal{T}$ is any congruence on Λ^2 including β-conversion $=_\beta$.

(ii) A λ-theory $\mathcal{T}$ is called:
- *consistent*, if $\mathcal{T}$ does not equate all λ-terms;
- *inconsistent*, if $\mathcal{T}$ is not consistent;
- *extensional*, if $\mathcal{T}$ contains the η-conversion $=_\eta$ as well;
- *sensible*, if $\mathcal{T}$ is consistent and equates all unsolvable λ-terms;
- *semi-sensible*, if $\mathcal{T}$ does not equate a solvable and an unsolvable.

We write $\mathcal{T} \vdash M = N$, or simply $M =_{\mathcal{T}} N$, whenever $(M, N) \in \mathcal{T}$.

The set of all λ-theories, ordered by inclusion, forms a quite rich complete lattice. We denote by $\boldsymbol{\lambda}$ (resp. $\boldsymbol{\lambda\eta}$) the smallest (resp. extensional) λ-theory. Both $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda\eta}$ are consistent, semi-sensible but not sensible. A λ-theory can be introduced syntactically, or semantically as the theory of a model. The model theory of λ-calculus is largely based on the notion of combinatory algebras, and its variations (see, e.g., [Koy82, Sel02, Mey82, HLS72] and [Bar84, Ch. 5]).

Definition 1.8.

(i) An *applicative structure* is given by $\mathcal{A} = (A, \cdot\,)$ where A is a set and $(\cdot)$ is a binary operation on A called *application*. We represent application as juxtaposition and we assume it is left-associative, e.g., $abc = (a \cdot b) \cdot c$. An equivalence $\simeq$ on $\mathcal{A}$ is a *congruence* if it is compatible w.r.t. application:
$$a \simeq a' \ \wedge\ b \simeq b' \quad \Rightarrow \quad ab \simeq a'b'$$

(ii) A *combinatory algebra* $\mathcal{C} = (C, \cdot, \mathbf{k}, \mathbf{s})$ is an applicative structure for a signature with two constants $\mathbf{k}, \mathbf{s}$, such that $\mathbf{k} \neq \mathbf{s}$ and $(\forall x, y, z \in C)$:
$$\mathbf{k}xy = x, \text{ and } \mathbf{s}xyz = xz(yz).$$
We say that $\mathcal{C}$ is *extensional* if the following holds:
$$\forall x.\forall y.(\forall z.(xz = yz) \Rightarrow x = y)$$

(iii) Given a combinatory algebra $\mathcal{C}$ and a congruence $\simeq$ on $(C, \cdot\,)$, define:
$$\mathcal{C}_\simeq = (C/_\simeq, \bullet_\simeq, \mathbf{k}_\simeq, \mathbf{s}_\simeq)$$
where
$$\begin{array}{rcl} [a]_\simeq \bullet_\simeq [b]_\simeq & = & [a \cdot b]_\simeq, \\ \mathbf{k}_\simeq & = & [\mathbf{k}]_\simeq \\ \mathbf{s}_\simeq & = & [\mathbf{s}]_\simeq. \end{array}$$
It is easy to check that if $\mathbf{k} \not\simeq \mathbf{s}$ then $\mathcal{C}_\simeq$ is a combinatory algebra.

We call $\mathbf{k}$ and $\mathbf{s}$ the *basic combinators*; the derived combinators $\mathbf{i}$ and $\boldsymbol{\varepsilon}$ are defined by $\mathbf{i} = \mathbf{skk}$ and $\boldsymbol{\varepsilon} = \mathbf{s}(\mathbf{ki})$. It is not difficult to verify that every combinatory algebra satisfies the identities $\mathbf{i}x = x$ and $\boldsymbol{\varepsilon}xy = xy$.

It is well-known that combinatory algebras are models of combinatory logic. A λ-term M can be interpreted in any combinatory algebra $\mathcal{C}$ by first translating M into a term X of combinatory logic, written $(M)_{\mathrm{CL}} = X$, and then interpreting the latter in $\mathcal{C}$. However, there might be β-convertible λ-terms M, N that are interpreted as distinguished elements of $\mathcal{C}$. For this reason, not all combinatory algebras are actually models of λ-calculus.

The axioms of an elementary subclass of combinatory algebras, called λ-*models*, were expressly chosen to make coherent the definition of interpretation of λ-terms (see [Bar84, Def. 5.2.1]). The *Meyer-Scott axiom* is the most important axiom in the definition of a λ-model. In the first-order language of combinatory algebras it becomes:

$$\forall x.\forall y\,.\,(\forall z\,.\,(xz = yz) \Rightarrow \boldsymbol{\varepsilon} x = \boldsymbol{\varepsilon} y).$$

The combinator $\boldsymbol{\varepsilon}$ becomes an inner choice operator, that makes coherent the interpretation of an abstraction λ-term.

1.3. **Syntactic λ-models.** The definition of a λ-model is difficult to handle in practice because the five Curry's axioms [Bar84, Thm. 5.2.5] are complicated to verify by hand. To prove that a certain combinatory algebra is actually a λ-model, it is preferable to exploit Hindley's (equivalent) notion of a syntactic λ-model. See, e.g., [Koy82].

The definition of syntactic λ-model in [Koy82] is general enough to interpret λ-terms possibly containing constants $\hat{a}$ representing elements a of a set A. We follow that tradition and denote by $\Lambda(A)$ the set of all λ-terms possibly containing constants from A, and we call them λA-*terms*. For instance, given $a \in A$, we have $M = \mathbf{I}(\lambda x.x\hat{a})\hat{b} \in \Lambda(A)$. All notions, notations and results from Subsection 1.1 extend to λA-terms without any problem. In particular, substitution is extended by setting $\hat{a}[N/x] = \hat{a}$, for all $a \in A$ and $N \in \Lambda(A)$. As an example, the λA-term M above reduces as follows: $M \to_\beta (\lambda x.x\hat{a})\hat{b} \to_\beta \hat{b}\hat{a} \in \Lambda(A)$. Observe that substitutions of variables by constants always permute, namely $M[\hat{a}/x][\hat{b}/y] = M[\hat{b}/y][\hat{a}/x]$, for all $a, b \in A$.

Given a set A, a *valuation in* A is any map $\rho : \mathrm{Var} \to A$. We write Val_A for the set of all valuations in A. Given $\rho \in \mathrm{Val}_A$ and $a \in A$, define:

$$(\rho[x := a])(y) = \begin{cases} a, & \text{if } x = y, \\ \rho(y), & \text{otherwise.} \end{cases}$$

Definition 1.9. A *syntactic* λ-*model* is a tuple $\mathcal{S} = (A, \cdot, [\![-]\!]_-)$ such that $(A, \cdot)$ is an applicative structure and the *interpretation function*

$$[\![-]\!]_- : \Lambda(A) \times \mathrm{Val}_A \to A$$

satisfies

(i) $[\![x]\!]_\rho = \rho(x)$, for all $x \in \mathrm{Var}$;
(ii) $[\![\hat{a}]\!]_\rho = a$, for all $a \in A$;
(iii) $[\![PQ]\!]_\rho = [\![P]\!]_\rho \cdot [\![Q]\!]_\rho$;
(iv) $[\![\lambda x.P]\!]_\rho \cdot a = [\![P]\!]_{\rho[x:=a]}$, for all $a \in A$;
(v) $\forall x \in \mathrm{FV}(M)\,.\,\rho(x) = \rho'(x) \quad \Rightarrow \quad [\![M]\!]_\rho = [\![M]\!]_{\rho'}$;
(vi) $\forall a \in A\,.\,[\![M]\!]_{\rho[x:=a]} = [\![N]\!]_{\rho[x:=a]} \quad \Rightarrow \quad [\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$.

If $M \in \Lambda^o$, then $[\![M]\!]_\rho$ is independent from the valuation ρ and we simply write $[\![M]\!]$.

We write $\mathcal{S} \models M = N$ if and only if $\forall \rho \in \mathrm{Val}_A\,.\,[\![M]\!]_\rho = [\![N]\!]_\rho$ holds. It is easy to check that $\boldsymbol{\lambda} \vdash M = N$ entails $\mathcal{S} \models M = N$.

The λ-*theory induced by* $\mathcal{S}$ is defined as follows:

$$\mathrm{Th}(\mathcal{S}) = \{M = N \mid \mathcal{S} \models M = N\}.$$

The precise correspondence between λ-models and syntactic λ-models is described in [Bar84], Theorem 5.3.6. For our purposes, it is enough to know that if $\mathcal{S}$ is a syntactic λ-model then $\mathcal{C}_{\mathcal{S}} = (A, \cdot, [\![\mathbf{K}]\!], [\![\mathbf{S}]\!])$ is a λ-model. We say that $\mathcal{S}$ is *extensional* whenever $\mathcal{C}_{\mathcal{S}}$ is extensional as a combinatory algebra. This holds iff $\mathrm{Th}(\mathcal{S})$ is extensional iff $\mathcal{S} \models \mathbf{I} = \mathbf{1}$.

2. Addressing Machines

In this section we introduce the notion of an *Addressing Machine*. We first provide some intuitions, then we proceed with the formal description of such machines. The general structure of an addressing machine is composed by two substructures:

- the *internal components*, organized as follows:
  - a finite number of *internal registers*;
  - an *internal program*.
- the *input-tape*.

As the name suggests, the addressing mechanism is central in this formalism. Each addressing machine is associated with an address, receives a list of addresses in its input-tape and is able to transfer the computation to another machine by calling its address, possibly extending its input-tape.

2.1. Tapes, Registers and Programs.

We consider fixed a countable set $\mathbb{A}$ of *addresses*, together with a constant $\varnothing \notin \mathbb{A}$ that we call "null" and that corresponds to an uninitialized register.

Definition 2.1. We let $\mathbb{A}_\varnothing = \mathbb{A} \cup \{\varnothing\}$.

(i) An $\mathbb{A}$-*valued tape* T is a finite (possibly empty) ordered list of addresses $T = [a_1, \dots, a_n]$ with $a_i \in \mathbb{A}$ for all $i \leq n$. We write $\mathbb{T}_\mathbb{A}$ for the set of all $\mathbb{A}$-valued tapes.

(ii) Let $a \in \mathbb{A}$ and $T, T' \in \mathbb{T}_\mathbb{A}$. 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.

(iii) Given an index $i \in \mathbb{N}$, an $\mathbb{A}_\varnothing$-valued *register* R_i is a memory-cell capable of storing either $\varnothing$ or an address $a \in \mathbb{A}$.

(iv) Given $\mathbb{A}_\varnothing$-valued registers $R_0, \dots, R_n$ for $n \geq 0$, an address $a \in \mathbb{A}$ and an index $i \in \mathbb{N}$, we write $\vec{R}[R_i := a]$ for the registers $\vec{R}$ where the value of R_i has been updated:

$$R_0, \dots, R_{i-1}, a, R_{i+1}, \dots, R_n$$

Notice that, whenever $i > n$, we assume that $\vec{R}[R_i := a] = \vec{R}$.

Addressing machines can be seen as having a RISC architecture, since their internal program is composed by only three instructions. We describe the effects of these basic operations on a machine having r internal registers $R_0, \dots, R_{r-1}$. Therefore, when we say "if an internal register R_i exists" we mean that the condition $0 \leq i < r$ is satisfied. In the following, $i, j, k \in \mathbb{N}$ correspond to indices of internal registers:

- `Load` i: corresponds to the action of reading the first element a from the input-tape T, and writing a on the internal register R_i. If the input-tape is empty then the machine remains stuck waiting for an input (however, this is not considered as an error state).

  The *precondition* to execute the operation is that the input-tape is non-empty, namely $T = a :: T'$; the *postconditions* are that R_i, if it exists, contains the address a and the input-tape of the machine becomes T'. If R_i does not exist, i.e. when $i \geq r$, the content of $\vec{R}$ remains unchanged (i.e., the input element a is read and subsequently thrown away).

- $k \leftarrow \texttt{App}(i,\, j)$: corresponds to the action of reading the contents of R_i and R_j, calling an external *application map* on the corresponding addresses a_1, a_2, and writing the result in the internal register R_k, if it exists.

  The *precondition* is that R_i, R_j exist and are initialized, i.e. $R_i, R_j \neq \varnothing$. The *postcondition* is that R_k, if it exists, contains the address of the machine of address a_1 whose input-tape has been extended with a_2. Otherwise the content of $\vec{R}$ remains unchanged.
- $\texttt{Call}\ i$: transfers the computation to the machine whose address is stored in R_i, extending its input-tape with the addresses that are left in T.

  The *precondition* is that R_i exists and is initialized. The *postcondition* is that the machine having the address stored in R_i is executed on the extended input-tape.

We define what is a syntactically valid program of this language, and introduce a decision procedure for verifying that the preconditions of each instruction are satisfied when it is executed. As we will see in Lemma 2.5, these properties are decidable and statically verifiable. As a consequence, addressing machines will never give rise to an error at run-time.

Definition 2.2.

(i) A *program* P is a finite list of instructions generated by the following grammar (where ε represents the empty string, and $i, j, k \in \mathbb{N}$):

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

In other words a program starts with a list of $\texttt{Load}$'s, continues with a list of $\texttt{App}$'s and possibly ends with a $\texttt{Call}$. Each of these lists may be empty, in particular the empty-program ε can be generated.

(ii) Given a program P, an $r \in \mathbb{N}$, and a set $\mathcal{I} \subseteq \{0, \ldots, r-1\}$ of indices (representing initialized registers), define $\mathcal{I} \models^r P$ as the least relation closed under the rules:

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

$$\frac{i \in \mathcal{I}}{\mathcal{I} \models^r \texttt{Call}\ i} \qquad \frac{\mathcal{I} \models^r \texttt{A} \quad i, j \in \mathcal{I} \quad k \geq r}{\mathcal{I} \models^r k \leftarrow \texttt{App}(i,\, j);\ \texttt{A}} \qquad \frac{\mathcal{I} \models^r \texttt{P} \quad i \geq r}{\mathcal{I} \models^r \texttt{Load}\ i;\ \texttt{P}}$$

(iii) Let $r \in \mathbb{N}$ and $\vec{R} = R_0, \ldots, R_{r-1}$ be $\mathbb{A}_\varnothing$-valued registers. We say that a program P is *valid with respect to* $\vec{R}$ whenever $\mathcal{R} \models^r P$ holds for

$$\mathcal{R} = \{ i \mid R_i \neq \varnothing \ \wedge \ 0 \leq i < r \} \tag{2.1}$$

Notice that the notion of a valid program is independent from the tape of a machine.

Examples 2.3. Consider addresses $a_1, a_2 \in \mathbb{A}$, as well as $\mathbb{A}_\varnothing$-valued registers $R_0 = \varnothing$, $R_1 = a_1, R_2 = a_2, R_3 = \varnothing$ (so $r = 4$). In this example, the set $\mathcal{R}$ of initialized registers as defined in (2.1) is $\mathcal{R} = \{1, 2\}$.

P_n	Program	$\mathcal{R} \models^4 P_n$
$P_0 =$	$\texttt{Load}\ 0; 2 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 2$	✓
$P_1 =$	$0 \leftarrow \texttt{App}(1,\ 2);\ 3 \leftarrow \texttt{App}(0,\ 2);\ \texttt{Call}\ 3$	✓
$P_2 =$	$\texttt{Load}\ 5;\ \texttt{Load}\ 0;\ \texttt{Call}\ 0$	✓
$P_3 =$	$\texttt{Load}\ 5;\ 5 \leftarrow \texttt{App}(1,\ 2);\ \texttt{Call}\ 2$	✓
$P_4 =$	$2 \leftarrow \texttt{App}(0,\ 1);\ \texttt{Call}\ 2$	✗
$P_5 =$	$\texttt{Load}\ 0;\ \texttt{Call}\ 3$	✗
$P_6 =$	$3 \leftarrow \texttt{App}(1,\ 2);\ \texttt{Call}\ 5$	✗

Above we use "5" as an index of an unexisting register. Notice that a program trying to update an unexisting register remains valid (see P_2, P_3), the new value is simply discharged. On the contrary, an attempt at reading the content of an uninitialized (P_4, P_5) or unexisting (P_6) register invalidates the whole program.

Notation 2.4. We use "$-$" to indicate an arbitrary index of an unexisting register. E.g., the program P_6 will be written $3 \leftarrow \texttt{App}(1,\ 2);\ \texttt{Call}\ -$. We also write $\texttt{Load}\ (i_1, \dots, i_k)$ as an abbreviation for $\texttt{Load}\ i_1; \cdots ; \texttt{Load}\ i_k;$. By employing all these notations, P_2 can be written as $P_2 = \texttt{Load}\ (-, 0); \texttt{Call}\ 0$.

Lemma 2.5. *For all $\mathbb{A}_\varnothing$-valued registers $\vec{R}$ and program P it is decidable whether P is valid with respect to $\vec{R}$.*

Proof. Decidability follows from the fact that the grammar in Definition 2.2(i) is right-linear, the list of registers $\vec{R}$ is finite, the rules in Definition 2.2(ii) are syntax-directed and their side conditions are decidable. □

2.2. **Addressing machines and their operational semantics.** Everything is in place to introduce the definition of an addressing machine. Thanks to Lemma 2.5 it is reasonable to require that an addressing machine has a valid internal program.

Definition 2.6. (i) An *addressing machine* M *(with r registers) over* $\mathbb{A}$ is given by a tuple:

$$\mathsf{M} = \langle \vec{R}, P, T \rangle$$

where:
- $\vec{R} = R_0, \dots, R_{r-1}$ are $\mathbb{A}_\varnothing$-valued registers;
- P is a program valid w.r.t. $\vec{R}$;
- T is an $\mathbb{A}$-valued *(input) tape*.

(ii) We write $\mathsf{M}.r$ for the number of registers of M, $\mathsf{M}.\vec{R}$ for the list of its registers, $\mathsf{M}.R_i$ for its i-th register, $\mathsf{M}.P$ for the associated program and finally $\mathsf{M}.T$ for its input tape.

(iii) We say that an addressing machine M as above is *stuck*, in symbols $\mathsf{stuck}(\mathsf{M})$, whenever its program has shape $\mathsf{M}.P = \texttt{Load}\ i; \mathsf{P}$ but its input-tape is empty $\mathsf{M}.T = []$. Otherwise, M is *not stuck*, in symbols: $\neg\mathsf{stuck}(\mathsf{M})$.

(iv) The set of all addressing machines over $\mathbb{A}$ will be denoted by $\mathcal{M}_{\mathbb{A}}$.

The machines below will be used as running examples in the next sections. Intuitively, the addressing machines $\mathsf{K}, \mathsf{S}, \mathsf{I}, \mathsf{D}, \mathsf{O}$ mimic the behavior of the λ-terms $\mathbf{K}$, $\mathbf{S}$, $\mathbf{I}$, $\boldsymbol{\Delta}$ and $\boldsymbol{\Omega}$, respectively. For writing their programs, we adopt the conventions introduced in Notation 2.4.

Examples 2.7. The following are addressing machines.

(i) For every $n \in \mathbb{N}$, define an addressing machine with $n+1$ registers as:

$$\mathsf{x}_n = \langle R_0, \dots, R_n, \varepsilon, [] \rangle, \text{ where } \vec{R} := \vec{\varnothing}.$$

We call $\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2, \dots$ *indeterminate machines* because they share some analogies with variables (they can be used as place holders).

(ii) The addressing machine K with 1 register R_0 is defined by:

$$\mathsf{K} = \langle \varnothing, \texttt{Load}\ (0, -); \texttt{Call}\ 0, [] \rangle$$

(iii) The addressing machine S with 3 registers is defined by:

$$\begin{array}{lcl} \mathsf{S} & = & \langle \varnothing, \varnothing, \varnothing, P, [] \rangle, \text{ where:} \\ \mathsf{S}.P & = & \texttt{Load}\ (0,1,2);\ 0 \leftarrow \texttt{App}(0,\ 2); \\ & & 1 \leftarrow \texttt{App}(1,\ 2);\ 2 \leftarrow \texttt{App}(0,\ 1);\ \texttt{Call}\ 2 \end{array}$$

(iv) Assume that $k \in \mathbb{A}$ represents the address associated with the addressing machine K. Define the addressing machine I as $\mathsf{I} = \langle \varnothing^3, \mathsf{S}.P, [k,k] \rangle$.

(v) The addressing machine D with 1 register is given by:

$$\mathsf{D} = \langle \varnothing, \texttt{Load}\ 0;\ 0 \leftarrow \texttt{App}(0,\ 0);\ \texttt{Call}\ 0, [] \rangle$$

(vi) Assume that $d \in \mathbb{A}$ represents the address of the addressing machine D. Define the addressing machine O by setting $\mathsf{O} = \langle \varnothing, \mathsf{D}.P, [d] \rangle$.

We now enter into the details of the addressing mechanism which constitutes the core of this formalism. In an implementation of addressing machines, it would be reasonable to pick up a fresh address from $\mathbb{A}$ whenever a new machine is constructed and save the correspondence in some address table. See Section 6 for more implementation details. To construct a λ-model, we need a uniform way of associating machines with their addresses.

Definition 2.8. Fix a bijective map $\# : \mathcal{M}_{\mathbb{A}} \to \mathbb{A}$ from the set of all addressing machines over $\mathbb{A}$ to the set $\mathbb{A}$ of addresses. We call the map $\#(\cdot)$ an *Address Table Map (ATM)*.

(i) Given $M \in \mathcal{M}_{\mathbb{A}}$, we say that $\#\mathsf{M}$ is the *address of* M.

(ii) Given an address $a \in \mathbb{A}$, we write $\#^{-1}(a)$ for the unique machine having address a. In other words, we have $\#^{-1}(a) = \mathsf{M} \iff \#\mathsf{M} = a$.

(iii) Given $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ and $T' \in \mathbb{T}_{\mathbb{A}}$, we write $\mathsf{M} \mathbin{@} T'$ for the machine

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

(iv) Define the *application map* $(\cdot) : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$ as follows

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

That is, the *application* of a to b is the unique address c of the addressing machine obtained by adding b at the end of the input tape of the addressing machine $\#^{-1}(a)$.

Since both $\mathcal{M}_{\mathbb{A}}$ and $\mathbb{A}$ are countable sets, there exist $2^{\aleph_0}$ possible choices for an ATM.

Remark 2.9. Depending on the chosen ATM $\#(-)$, there might exist addressing machines calling each other, as in $\mathsf{M} = \langle \#\mathsf{N}, \texttt{Call}\ 0, [] \rangle$ and $\mathsf{N} = \langle \#\mathsf{M}, \texttt{Call}\ 0, [] \rangle$, or even countably many machines $(\mathsf{M}_n)_{n \in \mathbb{N}}$ satisfying $\mathsf{M}_n = \langle \#\mathsf{M}_{n+1}, \varepsilon, [] \rangle$. Therefore, in general, the process of recursively dereferencing the addresses stored in the registers (or tape) of a machine might not terminate. This kind of behaviour is not pathological, rather intrinsic to the notions of addresses and dereference operators.

In practice, one may desire to work with an ATM performing the association between addressing machines and their addresses in a computable way. However, we do not require our ATMs to satisfy any effectiveness conditions since it would be peculiar to propose a model of computation depending on a pre-existing notion of "computable". The results presented in this paper are independent from the ATM under consideration.

Definition 2.10 (Small step operational semantics)**.** Define a reduction strategy on addressing machines representing one head-step of computation

$$\to_{\mathsf{h}} \subseteq \mathcal{M}_{\mathbb{A}} \to \mathcal{M}_{\mathbb{A}}$$

as the least relation closed under the following rules:

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

As usual, we write $\twoheadrightarrow_{\mathsf{h}}$ for the transitive-reflexive closure of $\to_{\mathsf{h}}$. We say that an addressing machine M *is in a final state* if there is no N such that $\mathsf{M} \to_{\mathsf{h}} \mathsf{N}$. We write $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}(\mathsf{N})$ whenever $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$ and $\mathsf{stuck}(\mathsf{N})$ hold. When N is not important, we simply write $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}()$. Similarly, $\mathsf{M} \not\twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}()$ means that M never reduces to a stuck addressing machine.

Remark 2.11.

(i) Definition 2.10 is well defined since the validity of a program is preserved by h-reduction: if $\mathsf{M} \to_{\mathsf{h}} \mathsf{N}$ and $\mathsf{M}.P$ is valid w.r.t. $\mathsf{M}.\vec{R}$ then $\mathsf{N}.P$ is valid w.r.t. $\mathsf{N}.\vec{R}$. This follows immediately from Definition 2.2(ii). In particular when executing $\mathtt{Load}\ i$, or $\mathtt{Call}\ i$, R_i must be initialized and when executing $k \leftarrow \mathtt{App}(i,\ j)$ we must have $R_i, R_j \neq \varnothing$.

(ii) Addressing machines in a final state are either of the form $\langle \vec{R}, \varepsilon, T\rangle$ or $\langle \vec{R}, \mathtt{Load}\ i; P, []\rangle$, and in the latter case they are stuck.

Lemma 2.12. *The reduction strategy* $\to_{\mathsf{h}}$ *enjoys the following properties:*

(i) *Determinism:* $\mathsf{M} \to_{\mathsf{h}} \mathsf{N}_1 \ \wedge\ \mathsf{M} \to_{\mathsf{h}} \mathsf{N}_2 \ \Rightarrow\ \mathsf{N}_1 = \mathsf{N}_2$*.*

(ii) *Closure under application:* $\forall a \in \mathbb{A}\,.\, \mathsf{M} \to_{\mathsf{h}} \mathsf{N} \ \Rightarrow\ \mathsf{M} \mathbin{@} [a] \ \to_{\mathsf{h}} \mathsf{N} \mathbin{@} [a]\,.$

Proof. (i) Since the applicable rule from Definition 2.10, if any, is uniquely determined by the first instruction on $\mathsf{M}.P$ and its input-tape $\mathsf{M}.T$.

(ii) Easy. By cases on the rule applied for deriving $\mathsf{M} \to_{\mathsf{h}} \mathsf{N}$. □

Examples 2.13. For brevity, we sometimes display only the first instruction of the internal program. Take $a, b, c \in \mathbb{A}$.

(i) We show that K behaves as the first projection:

$$\begin{array}{rcl} \mathsf{K} \mathbin{@} [a, b] & = & \langle \varnothing, \mathtt{Load}\ (0, -); \mathtt{Call}\ 0, [a, b]\rangle \\ & \to_{\mathsf{h}} & \langle a, \mathtt{Load}\ -; \mathtt{Call}\ 0, [b]\rangle \to_{\mathsf{h}} \langle a, \mathtt{Call}\ 0, []\rangle \to_{\mathsf{h}} \#^{-1}(a). \end{array}$$

(ii) We verify that S behaves as the combinator $\mathbf{S}$ from combinatory logic:

$$\begin{array}{rcl} \mathsf{S} \mathbin{@} [a, b, c] & = & \langle \varnothing^3, \mathtt{Load}\ (0, 1, 2); \cdots, [a, b, c]\rangle \\ & \twoheadrightarrow_{\mathsf{h}} & \langle a, b, c, 0 \leftarrow \mathtt{App}(0,\ 2); \cdots, []\rangle \\ & \to_{\mathsf{h}} & \langle a \cdot c, b, c, 1 \leftarrow \mathtt{App}(1,\ 2); \cdots, []\rangle \\ & \to_{\mathsf{h}} & \langle a \cdot c, b \cdot c, c, 2 \leftarrow \mathtt{App}(0,\ 1); \cdots, []\rangle \\ & \to_{\mathsf{h}} & \langle a \cdot c, b \cdot c, (a \cdot c) \cdot (b \cdot c), \mathtt{Call}\ 2; \cdots, []\rangle \\ & \to_{\mathsf{h}} & \#^{-1}((a \cdot c) \cdot (b \cdot c)) \end{array}$$

(iii) As expected, $\mathsf{I} = \mathsf{S} \mathbin{@} [\#\mathsf{K}, \#\mathsf{K}]$ behaves as the identity:

$$\begin{array}{rcl}
\mathsf{I} \mathbin{@} [a] & = & \langle \varnothing^3, \texttt{Load } (0,1,2); \cdots, [\#\mathsf{K}, \#\mathsf{K}, a] \rangle \\
& \twoheadrightarrow_{\mathsf{h}} & \langle \#\mathsf{K}, \#\mathsf{K}, a, 0 \leftarrow \texttt{App}(0,\ 2); \cdots, [] \rangle \\
& \to_{\mathsf{h}} & \langle \#\mathsf{K} \cdot a, \#\mathsf{K}, a, 1 \leftarrow \texttt{App}(1,\ 2); \cdots, [] \rangle \\
& \to_{\mathsf{h}} & \langle \#\mathsf{K} \cdot a, \#\mathsf{K} \cdot a, a, 2 \leftarrow \texttt{App}(0,\ 1); \cdots, [] \rangle \\
& \to_{\mathsf{h}} & \langle \#\mathsf{K} \cdot a, \#\mathsf{K} \cdot a, \#\mathsf{K} \cdot a \cdot (\#\mathsf{K} \cdot a), \texttt{Call } 2; [] \rangle \\
& \to_{\mathsf{h}} & \mathsf{K} \mathbin{@} [a, \#\mathsf{K} \cdot a] \\
& = & \langle \varnothing, \texttt{Load } (0, -); \cdots, [a, \#\mathsf{K} \cdot a] \rangle \\
& \twoheadrightarrow_{\mathsf{h}} & \langle a, \#\mathsf{K} \cdot a, \texttt{Call } 0, [] \rangle \to_{\mathsf{h}} \#^{-1}(a)
\end{array}$$

(iv) Finally, we check that O gives rise to an infinite reduction sequence:

$$\begin{array}{rcl}
\mathsf{O} & = & \langle \varnothing, \texttt{Load } 0;\ 0 \leftarrow \texttt{App}(0,\ 0);\ \texttt{Call } 0, [\#\mathsf{D}] \rangle \\
& \to_{\mathsf{h}} & \langle \#\mathsf{D}, 0 \leftarrow \texttt{App}(0,\ 0);\ \texttt{Call } 0, [] \rangle \\
& \to_{\mathsf{h}} & \langle \#(\mathsf{D} \mathbin{@} [\#\mathsf{D}]), \texttt{Call } 0, [] \rangle \to_{\mathsf{h}} \mathsf{D} \mathbin{@} [\#\mathsf{D}] = \mathsf{O} \twoheadrightarrow_{\mathsf{h}} \cdots
\end{array}$$

Similarly, we can define a big-step operational semantics relating an addressing machine M with its final result (if any).

Definition 2.14 (Big-step semantics)**.** Define $\mathsf{M} \Downarrow \mathsf{V}$, where $\mathsf{M}, \mathsf{V} \in \mathcal{M}_{\mathbb{A}}$ and V is in a final state, as the least relation closed under the following rules:

$$\frac{\mathsf{M}.P = \texttt{Load } i; P' \quad \mathsf{M}.T = []}{\mathsf{M} \Downarrow \mathsf{M}}\ (\text{Stuck}) \qquad\qquad \frac{\mathsf{M}.P = \varepsilon}{\mathsf{M} \Downarrow \mathsf{M}}\ (\text{End})$$

$$\frac{\mathsf{M}.P = \texttt{Load } i; P' \quad \mathsf{M}.T = a :: T' \quad \langle \mathsf{M}.\vec{R}[R_i := a], P', T' \rangle \Downarrow \mathsf{V}}{\mathsf{M} \Downarrow \mathsf{V}}\ (\text{Load})$$

$$\frac{\mathsf{M}.P = k \leftarrow \texttt{App}(i,\ j); P' \quad a = \mathsf{M}.R_i \cdot \mathsf{M}.R_j \quad \langle \mathsf{M}.\vec{R}[R_k := a], \mathsf{M}.P', \mathsf{M}.T \rangle \Downarrow \mathsf{V}}{\mathsf{M} \Downarrow \mathsf{V}}\ (\text{App})$$

$$\frac{\mathsf{M}.P = \texttt{Call } i \quad \mathsf{M}' = \#^{-1}(\mathsf{M}.R_i) \quad \mathsf{M}' \mathbin{@} [\mathsf{M}.T] \Downarrow \mathsf{V}}{\mathsf{M} \Downarrow \mathsf{V}}\ (\text{Call})$$

Example 2.15. Recall that $\mathsf{K}.P = \texttt{Load } (0, -);\ \texttt{Call } 0$. Notice that we cannot prove $\mathsf{K} \mathbin{@} [a, b] \Downarrow \#^{-1}(a)$ for an arbitrary $a \in \mathbb{A}$, as we need to ensure that the resulting machine is in a final state. For this reason, we will use indeterminate machines $\mathsf{x}_1, \mathsf{x}_2$ from Example 2.7(i).

$$\frac{\mathsf{K}.P = \texttt{Load } 0;\ P'; \qquad \dfrac{P' = \texttt{Load } -; P'' \qquad \dfrac{P'' = \texttt{Call } 0 \quad R_0 = \#\mathsf{x}_1 \quad \dfrac{(\text{End})}{\mathsf{x}_1 \Downarrow \mathsf{x}_1}}{\langle \#\mathsf{x}_1, P'', [] \rangle \Downarrow \mathsf{x}_1}}{\langle \#\mathsf{x}_1, P', [\#\mathsf{x}_2] \rangle \Downarrow \mathsf{x}_1}}{\mathsf{K} \mathbin{@} [\#\mathsf{x}_1, \#\mathsf{x}_2] \Downarrow \mathsf{x}_1}$$

We now show that the two operational semantics are equivalent on terminating computations.

Proposition 2.16. *For $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$, the following are equivalent:*

(1) $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{N} \not\to_{\mathsf{h}}$;
(2) $\mathsf{M} \Downarrow \mathsf{N}$.

Proof. (1 ⇒ 2) By induction on the length n of the reduction $\mathsf{M} = \mathsf{M}_1 \to_{\mathsf{h}} \mathsf{M}_2 \to_{\mathsf{h}} \cdots \to_{\mathsf{h}} \mathsf{M}_n = \mathsf{N} \not\to_{\mathsf{h}}$.

Case $n = 0$. By assumption N is in a final state. By Remark 2.11(ii), it is either of the form $\mathsf{N} = \langle \vec{R}, \varepsilon, T\rangle$ or it is stuck $\mathsf{N} = \langle \vec{R}, \texttt{Load}\ i; P, []\rangle$. In the former case we apply (End), in the latter (Stuck).

Case $n > 1$. Since $\mathsf{M}_1 \to_{\mathsf{h}} \mathsf{M}_2$, we have $\mathsf{M}_1.P \neq \varepsilon$. As the length of $\mathsf{M}_2 \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$ is $n-1$, by induction hypothesis we have a derivation of $\mathsf{M}_2 \Downarrow \mathsf{N}$. Depending on the first instruction in $\mathsf{M}_1.P$, we use this derivation to apply the homonymous rule (Load), (App) or (Call) and derive $\mathsf{M} \Downarrow \mathsf{N}$.

($2 \Rightarrow 1$) By induction on a derivation of $\mathsf{M} \Downarrow \mathsf{N}$.

Cases (Stuck) or (End). Then, $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{M} = \mathsf{N}$ by reflexivity of $\twoheadrightarrow_{\mathsf{h}}$.

Case (Load), i.e. $\mathsf{M}.P = \texttt{Load}\ i; P'$. In this case, we have that $\mathsf{M} \to_{\mathsf{h}} \langle \mathsf{M}.\vec{R}[R_i := a], P', \mathsf{M}.T\rangle \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$, by induction hypothesis.

Case (App), i.e. $\mathsf{M}.P = k \leftarrow \texttt{App}(i,\ j); P'$. Let us call $a = \mathsf{M}.R_j \cdot \mathsf{M}.R_k$. Then we have $\mathsf{M} \to_{\mathsf{h}} \langle \mathsf{M}.\vec{R}[R_k := a], P', \mathsf{M}.T\rangle \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$, by induction hypothesis.

Case (Call), i.e. $\mathsf{M}.P = \texttt{Call}\ i$. In this case $\mathsf{M} \to_{\mathsf{h}} \mathsf{M}' \,@\, [\mathsf{M}.T]$ for $\mathsf{M}' = \#^{-1}(\mathsf{M}.R_i)$. By induction hypothesis $\mathsf{M}' \,@\, [\mathsf{M}.T] \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$, whence $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$. □

3. Combinatory Algebras via Evaluation Equivalence

In this section we show how to construct a combinatory algebra based on the addressing machines formalism. Recall that the addressing machines K and S have been defined in Example 2.7. Consider the algebraic structure

$$\mathcal{A} = (\mathbb{A}, \cdot, \#\mathsf{K}, \#\mathsf{S})$$

Since the application $(\cdot)$ is total, $\mathcal{A}$ is an applicative structure. However, it is *not* a combinatory algebra. For instance, the $\lambda\mathbb{A}$-term $\mathbf{K}\hat{a}\hat{b}$ is interpreted as the address of the machine $\mathsf{K} \,@\, [a, b]$, which is *a priori* different from the address "a" because no computation is involved. Therefore, we need to quotient the algebra $\mathcal{A}$ by an equivalence relation equating at least all addresses corresponding to the same machine at different stages of the execution.

In the following, we denote by $\equiv_{\mathrm{R}}$ an arbitrary binary relation on $\mathcal{M}_{\mathbb{A}}$. The symbol R has no formal meaning, it is simply evocative of a relation. In the next definition, we are going to associate with every $\equiv_{\mathrm{R}}$ two relations, respectively denoted $\simeq_{\mathrm{R}} \subseteq \mathbb{A}^2$ and $=_{\mathrm{R}} \subseteq \mathcal{M}_{\mathbb{A}}^2$.

Definition 3.1. Every binary relation $\equiv_{\mathrm{R}} \subseteq \mathcal{M}_{\mathbb{A}}^2$ on addressing machines induces a relation $\simeq_{\mathrm{R}} \subseteq \mathbb{A}^2$ defined by

$$a \simeq_{\mathrm{R}} b \iff \#^{-1}(a) \equiv_{\mathrm{R}} \#^{-1}(b)$$

which is then extended to:

(i) $\mathbb{A}_{\varnothing}$-valued registers:

$$R \simeq_{\mathrm{R}} R' \iff (R = \varnothing = R') \vee (R = a \simeq_{\mathrm{R}} b = R');$$

(ii) Tuples:

$$a_1, \dots, a_n \simeq_{\mathrm{R}} b_1, \dots, b_m \iff (n = m) \wedge (\forall i \in \{1, \dots, n\} \,.\, a_i \simeq_{\mathrm{R}} b_i);$$

(This also applies to tuples of $\mathbb{A}_{\varnothing}$-valued registers $\vec{R} \simeq_{\mathrm{R}} \vec{R}'$.)

(iii) $\mathbb{A}$-valued tapes:

$$[a_1, \dots, a_n] \simeq_{\mathrm{R}} [b_1, \dots, b_m] \iff \vec{a} \simeq_{\mathrm{R}} \vec{b} \text{ (seen as tuples).}$$

In its turn, $\simeq_{\mathrm{R}}$ induces a relation $=_{\mathrm{R}} \subseteq \mathcal{M}_{\mathbb{A}}^2$ defined by setting (for all machines $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$):

$$\mathsf{M} =_{\mathrm{R}} \mathsf{N} \iff (\mathsf{M}.\vec{R} \simeq_{\mathrm{R}} \mathsf{N}.\vec{R}) \wedge (\mathsf{M}.P = \mathsf{N}.P) \wedge (\mathsf{M}.T \simeq_{\mathrm{R}} \mathsf{N}.T)$$

In particular, $\mathsf{M} =_{\mathrm{R}} \mathsf{N}$ entails that M and N share the same internal program, the number of internal registers, and the length of their input tape.

Lemma 3.2. *If the relation $\equiv_{\mathrm{R}}$ is an equivalence then so are $\simeq_{\mathrm{R}}$ and $=_{\mathrm{R}}$.*

Proof. Assume that $\equiv_{\mathrm{R}}$ is an equivalence. Then, the fact that $\simeq_{\mathrm{R}}$ is an equivalence follows from its definition since $\#^{-1}(\cdot)$ is a bijection. Concerning the relation $=_{\mathrm{R}}$, reflexivity, symmetry and transitivity follow immediately from the same properties of $\simeq_{\mathrm{R}}$ and $=$. □

Definition 3.3. Define $\equiv_{\mathbb{A}} \subseteq \mathcal{M}_{\mathbb{A}}^2$ as the least equivalence closed under:

$$\frac{\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =_{\mathbb{A}} \mathsf{N}}{\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}}\ (\twoheadrightarrow_{\mathbb{A}})$$

We say that M, N are *evaluation equivalent* whenever $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.

Remark 3.4.
(i) Reflexivity can be treated as a special case of the rule $(\twoheadrightarrow_{\mathbb{A}})$ since $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{M} =_{\mathbb{A}} \mathsf{M}$.
(ii) It follows from the definition that $=_{\mathbb{A}} \subseteq \equiv_{\mathbb{A}}$ and that $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$ entails $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.

Examples 3.5. From the calculations in Examples 2.13, it follows that

$$\begin{array}{lll} \mathsf{K} @ [\#\mathsf{x}_1, \#\mathsf{x}_2] & \equiv_{\mathbb{A}} & \mathsf{x}_1, \\ \mathsf{S} @ [\#\mathsf{x}_1, \#\mathsf{x}_2, \#\mathsf{x}_3] & \equiv_{\mathbb{A}} & (\mathsf{x}_1 @ [\#\mathsf{x}_3]) @ [\#(\mathsf{x}_2 @ [\#\mathsf{x}_3])]. \end{array}$$

Lemma 3.6. *The relation $\simeq_{\mathbb{A}}$ is a congruence on $\mathcal{A} = (\mathbb{A}, \cdot, \#\mathsf{K}, \#\mathsf{S})$.*

Proof. By definition $\equiv_{\mathbb{A}}$ is an equivalence, whence so is $\simeq_{\mathbb{A}}$ by Lemma 3.2. Let us check that $\simeq_{\mathbb{A}}$ is compatible w.r.t. $(\cdot)$. Consider $a \simeq_{\mathbb{A}} a'$ and $b \simeq_{\mathbb{A}} b'$. Call $\mathsf{M} = \#^{-1}(a)$ and $\mathsf{N} = \#^{-1}(a')$ and proceed by induction on a derivation of $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$, splitting into cases depending on the last applied rule.

$(\twoheadrightarrow_{\mathbb{A}})$ By definition, there exists $\mathsf{Z} \in \mathcal{M}_{\mathbb{A}}$ such that $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =_{\mathbb{A}} \mathsf{N}$. By Lemma 2.12(ii), $\mathsf{M} @ [b] \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} @ [b] =_{\mathbb{A}} \mathsf{N} @ [b']$ whence $a \cdot b \simeq_{\mathbb{A}} a' \cdot b'$.

(Transitivity) and (Symmetry) follow from the induction hypothesis. □

In order to prove that the congruence $\simeq_{\mathbb{A}}$ is non-trivial, we are going to characterize the equivalence $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$ it in terms of confluent reductions. For this purpose, we extend $\to_{\mathsf{h}}$ in such a way that reductions are also possible within registers and elements of the input-tape of an addressing machine.

Definition 3.7. Define the reduction relation $\to_{\mathsf{c}} \subseteq \mathcal{M}_{\mathbb{A}}^2$ as the least relation containing $\to_{\mathsf{h}}$ and closed under the following rules:

$$\frac{R_i = a \in \mathbb{A} \quad 0 \le i < r \quad \#^{-1}(a) \to_{\mathsf{c}} \mathsf{M}}{\langle R_0, \dots, R_{r-1}, P, T \rangle \to_{\mathsf{c}} \langle \vec{R}[R_i := \#\mathsf{M}], P, T \rangle}\ (\to_{\mathsf{i}}^{R})$$

$$\frac{0 \le i \le n \quad \#^{-1}(a_i) \to_{\mathsf{c}} \mathsf{M}}{\langle \vec{R}, P, [a_0, \dots, a_n] \rangle \to_{\mathsf{c}} \langle \vec{R}, P, [a_0, \dots, a_{i-1}, \#\mathsf{M}, a_{i+1}, \dots, a_n] \rangle}\ (\to_{\mathsf{i}}^{T})$$

We write $\mathsf{M} \to_{\mathsf{i}} \mathsf{N}$ if N is obtained from M by directly applying one of the above rules — this is called an *inner* step of computation. The transitive and reflexive closure of $\to_{\mathsf{c}}$ and $\to_{\mathsf{i}}$ are denoted by $\twoheadrightarrow_{\mathsf{c}}$ and $\twoheadrightarrow_{\mathsf{i}}$, respectively.

Lemma 3.8 (Postponement of inner steps)**.**
For $\mathsf{M}, \mathsf{N}, \mathsf{N}' \in \mathcal{M}_{\mathbb{A}}$*, if* $\mathsf{M} \to_{\mathsf{i}} \mathsf{N} \to_{\mathsf{h}} \mathsf{N}'$ *then there exists* $\mathsf{M}' \in \mathcal{M}_{\mathbb{A}}$ *such that* $\mathsf{M} \to_{\mathsf{h}} \mathsf{M}' \twoheadrightarrow_{\mathsf{i}} \mathsf{N}'$*. In diagrammatic form:*

$$\begin{tikzcd} \mathsf{M} \arrow[r, "\mathsf{i}"] \arrow[d, dashed, "\mathsf{h}"] & \mathsf{N} \arrow[d, "\mathsf{h}"] \\ \mathsf{M}' \arrow[r, dashed, two heads, "\mathsf{i}"] & \mathsf{N}' \end{tikzcd}$$

Proof. By cases analysis over $\mathsf{M} \to_{\mathsf{i}} \mathsf{N}$. The only interesting case is when the contracted redex is duplicated in $\mathsf{N} \to_{\mathsf{h}} \mathsf{N}'$, namely:

Case $\mathsf{M} = \langle \vec{R}[R_i := a], P, T\rangle$, $\mathsf{N} = \langle \vec{R}[R_i := b], P, T\rangle$ with $\mathsf{M}.P = \mathsf{N}.P = k \leftarrow \mathtt{App}(i,\, j); P'$ and $\#^{-1}(a) \to_{\mathsf{c}} \#^{-1}(b)$. Assume $i \neq k < \mathsf{M}.r$ and $i = j$, the other cases being easier. In this case $\mathsf{M}' = \langle \vec{R}[R_i := a][R_k := a \cdot a], P, T\rangle$, therefore we need 3 inner steps to close the diagram:

$$\begin{array}{rll} \mathsf{M}' & \to_{\mathsf{i}} & \langle \vec{R}[R_i := b][R_k := a \cdot a], P, T\rangle \\ & \to_{\mathsf{i}} & \langle \vec{R}[R_i := b][R_k := b \cdot a], P, T\rangle \\ & \to_{\mathsf{i}} & \langle \vec{R}[R_i := b][R_k := b \cdot b], P, T\rangle = \mathsf{N}'. \end{array}$$

This concludes the proof. □

Morally, the term rewriting system $(\mathcal{M}_{\mathbb{A}}, \to_{\mathsf{c}})$ is orthogonal because (i) the reduction rules defining $\to_{\mathsf{c}}$ are non-overlapping as $\to_{\mathsf{h}}$ is deterministic, $(\to_{\mathsf{i}}^{R})$ reduces a register and $(\to_{\mathsf{i}}^{T})$ reduces one element of the tape; (ii) the terms on the left-hand side of the arrow are linear, as no equality among subterms is required. Now, it is well-known that orthogonal TRS are confluent, but one cannot apply [Ter03, Thm.4.3.4] directly since we are not exactly dealing with first-order terms (because of the presence of the encoding).

Proposition 3.9. *The reduction* $\to_{\mathsf{c}}$ *is confluent.*

Proof sketch. The Parallel Moves Lemma, which is the key property for proving Theorem 4.3.4 in [Ter03] generalizes easily. The rest of the proof follows. □

Lemma 3.10. *Let* $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$*.*

(i) $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$ *entails* $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.
(ii) $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ *entails* $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.

Proof. (i) By induction on a derivation of $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$.

Base case $\mathsf{M} \to_{\mathsf{h}} \mathsf{N}$. Since $\equiv_{\mathbb{A}}$ is an equivalence then so is $=_{\mathbb{A}}$, by Lemma 3.2. In particular $=_{\mathbb{A}}$ is reflexive, whence $\mathsf{N} =_{\mathbb{A}} \mathsf{N}$. By Definition 3.3, we obtain $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.

Case $(\to_{\mathsf{i}}^{R})$. Then $\mathsf{M} = \langle \vec{R}[R_i := \#\mathsf{M}'], P, T\rangle$ and $\mathsf{N} = \langle \vec{R}[R_i := \#\mathsf{N}'], P, T\rangle$ for some existing register R_i and $\mathsf{M}', \mathsf{N}' \in \mathcal{M}_{\mathbb{A}}$ such that $\mathsf{M}' \to_{\mathsf{c}} \mathsf{N}'$. By induction hypothesis we get $\mathsf{M}' \equiv_{\mathbb{A}} \mathsf{N}'$, equivalently $\#\mathsf{M}' \simeq_{\mathbb{A}} \#\mathsf{N}'$. From this and reflexivity, it follows $\vec{R}[R_i := \#\mathsf{M}'] \simeq_{\mathbb{A}} \vec{R}[R_i := \#\mathsf{N}']$, $P \simeq_{\mathbb{A}} P$ and $T \simeq_{\mathbb{A}} T$. Thus $\mathsf{M} =_{\mathbb{A}} \mathsf{N}$, so we conclude because $=_{\mathbb{A}} \subseteq \equiv_{\mathbb{A}}$.

Case $(\to_{\mathsf{i}}^{T})$. In this case, we have

$$\begin{array}{rcl} \mathsf{M} & = & \langle \vec{R}, P, [a_0, \ldots, a_{i-1}, \#\mathsf{M}', a_{i+1} \ldots, a_n]\rangle \\ \mathsf{N} & = & \langle \vec{R}, P, [a_0, \ldots, a_{i-1}, \#\mathsf{N}', a_{i+1} \ldots, a_n]\rangle \end{array}$$

with $\mathsf{M}' \to_{\mathsf{c}} \mathsf{N}'$. By induction hypothesis we get $\mathsf{M}' \equiv_{\mathbb{A}} \mathsf{N}'$, equivalently $\#\mathsf{M}' \simeq_{\mathbb{A}} \#\mathsf{N}'$. This entails $\mathsf{M}.T \simeq_{\mathbb{A}} \mathsf{N}.T$, from which it follows $\mathsf{M} =_{\mathbb{A}} \mathsf{N}$. Conclude as above.

(ii) By induction on the length n of the reduction $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$.

Case $n = 0$. Then $\mathsf{M} = \mathsf{N}$, so we get $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$ by reflexivity.

Case $n > 0$. Then $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}' \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$. By (i), we get $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{M}'$. Since the reduction $\mathsf{M}' \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ is strictly shorter, the induction hypothesis gives $\mathsf{M}' \equiv_{\mathbb{A}} \mathsf{N}$. Conclude by transitivity. □

Theorem 3.11. *For* $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$*, we have:*

$$\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N} \iff \exists \mathsf{Z} \in \mathcal{M}_{\mathbb{A}} \,.\, \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z} \;{}_{\mathsf{c}}\!\twoheadleftarrow \mathsf{N}$$

Proof. ($\Rightarrow$) By induction on a derivation of $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.

($\twoheadrightarrow_{\mathbb{A}}$) Assume that $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =_{\mathbb{A}} \mathsf{N}$. From $\mathsf{Z} =_{\mathbb{A}} \mathsf{N}$ we get that $\mathsf{Z}.r = \mathsf{N}.r$, $\mathsf{Z}.\vec{R} \simeq_{\mathbb{A}} \mathsf{N}.\vec{R}$, $\mathsf{Z}.P = \mathsf{N}.P$ and $\mathsf{Z}.T \simeq_{\mathbb{A}} \mathsf{N}.T$. Note that $\mathsf{Z}.R_i = \varnothing$ iff $\mathsf{N}.R_i = \varnothing$. Let us call $\mathcal{R}$ the set of indices i of, say, Z such that $\mathsf{Z}.R_i \neq \varnothing$. By assumption, for every $i \in \mathcal{R}$, we have $\mathsf{Z}.R_i = a_i, \mathsf{N}.R_i = a'_i$ for $a_i \simeq_{\mathbb{A}} a'_i$. Equivalently, $\#^{-1}(a_i) \equiv_{\mathbb{A}} \#^{-1}(a'_i)$ holds and its derivation is smaller than $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$. By induction hypothesis, they have a common reduct $\#^{-1}(a_i) \twoheadrightarrow_{\mathsf{c}} \mathsf{X}_i \;{}_{\mathsf{c}}\!\twoheadleftarrow \#^{-1}(a'_i)$. Similarly, calling $\mathsf{Z}.T = [b_1, \dots, b_n]$ and $\mathsf{N}.T = [b'_1, \dots, b'_m]$ we must have $m = n$ and $b_j \simeq_{\mathbb{A}} b'_j$ whence the induction hypothesis gives a common reduct $\#^{-1}(b_j) \twoheadrightarrow_{\mathsf{c}} \mathsf{Y}_j \;{}_{\mathsf{c}}\!\twoheadleftarrow \#^{-1}(b'_j)$. Putting all reductions together, we conclude:

$$\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} \twoheadrightarrow_{\mathsf{c}} \langle \mathsf{Z}.\vec{R}[R_i := \#\mathsf{X}_i]_{i\in\mathcal{R}}, \mathsf{Z}.P, [\#\mathsf{Y}_1, \dots, \#\mathsf{Y}_n]\rangle \;{}_{\mathsf{c}}\!\twoheadleftarrow \mathsf{N}$$

(Transitivity) By induction hypothesis and confluence (Proposition 3.9).

(Symmetry) Straightforward from the induction hypothesis.

($\Leftarrow$) By Lemma 3.10(ii) we get $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{Z}$ and $\mathsf{N} \equiv_{\mathbb{A}} \mathsf{Z}$, so we conclude by symmetry and transitivity. □

Proposition 3.12. $\mathcal{A}_{\simeq_{\mathbb{A}}}$ *is a non-extensional combinatory algebra.*

Proof. From the calculations in Example 3.5, it follows that $\#\mathsf{K} \cdot a \cdot b \simeq_{\mathbb{A}} a$ and $\#\mathsf{S} \cdot a \cdot b \cdot c \simeq_{\mathbb{A}} (a \cdot c) \cdot (b \cdot c)$ hold, for all $a, b, c \in \mathbb{A}$. Notice that both addressing machines K and S are stuck, and $\mathsf{K} \neq_{\mathbb{A}} \mathsf{S}$ since, e.g., $\mathsf{K}.r \neq \mathsf{S}.r$. By Theorem 3.11, we get $\#\mathsf{K} \not\simeq_{\mathbb{A}} \#\mathsf{S}$, whence $\mathcal{A}_{\simeq_{\mathbb{A}}}$ is a combinatory algebra.

To check that $\mathcal{A}_{\simeq_{\mathbb{A}}}$ is not extensional, it is sufficient to exhibit two elements of $\mathbb{A}$ that are extensionally equal, but distinguished modulo $\simeq_{\mathbb{A}}$. For instance, take $\#\mathsf{K} \cdot a$ and $\#\mathsf{K}' \cdot a$, where $a \in \mathbb{A}$ is arbitrary and K' is a different implementation of the combinator $\mathbf{K}$, namely:

$$\begin{array}{lcl} \mathsf{K}' & = & \langle \varnothing, \varnothing, \mathtt{Load}\ (0,1); \mathtt{Call}\ 0, []\rangle, \\ \mathsf{K} & = & \langle \varnothing, \mathtt{Load}\ (0,-); \mathtt{Call}\ 0, []\rangle. \end{array}$$

For all $a, b \in \mathbb{A}$, easy calculations give $\#\mathsf{K}' \cdot a \cdot b \simeq_{\mathbb{A}} a$. Thus, for all $b \in \mathbb{A}$, we have

$$\#\mathsf{K} \cdot a \cdot b \simeq_{\mathbb{A}} a \simeq_{\mathbb{A}} \#\mathsf{K}' \cdot a \cdot b,$$

whence the two addresses $\#\mathsf{K} \cdot a$ and $\#\mathsf{K}' \cdot a$ are extensionally equal elements of $\mathcal{A}_{\simeq_{\mathbb{A}}}$. However, the corresponding addressing machines are both stuck and $\mathsf{K} @ [a] \neq_{\mathbb{A}} \mathsf{K}' @ [a]$, because $1 = (\mathsf{K} @ [a]).r \neq (\mathsf{K}' @ [a]).r = 2$. Since they cannot have a common reduct, we derive $\mathsf{K} @ [a] \not\equiv_{\mathbb{A}} \mathsf{K}' @ [a]$ by Theorem 3.11. We conclude that $\#\mathsf{K} \cdot a \not\simeq_{\mathbb{A}} \#\mathsf{K}' \cdot a$. □

Lemma 3.13. *The combinatory algebra* $\mathcal{A}_{\simeq_{\mathbb{A}}}$ *is not a* λ*-model.*

Proof. We need to find $M, N \in \Lambda$ satisfying $M =_\beta N$, while $\mathcal{A}_{\simeq_{\mathbb{A}}} \not\models M = N$. Take $M = \lambda z.(\lambda x.x)z =_{\mathrm{CL}} \mathbf{S}(\mathbf{KI})\mathbf{I}$ and $N = \lambda x.x =_{\mathrm{CL}} \mathbf{I}$ where $\mathbf{I} = \mathbf{SKK}$.

Recall that $\mathsf{I} = \mathsf{S} @ [\#\mathsf{K}, \#\mathsf{K}]$. Easy calculations give:

$$\begin{array}{rcl} \mathsf{S} @ [\#\mathsf{K} \cdot \#\mathsf{I}, \#\mathsf{I}] & = & \langle \varnothing, \varnothing, \varnothing, \mathtt{Load}\ 0; \cdots, [\#\mathsf{K} \cdot \#\mathsf{I}, \#\mathsf{I}]\rangle \\ & \to_{\mathsf{h}} & \langle \#\mathsf{K} \cdot \#\mathsf{I}, \varnothing, \varnothing, \mathtt{Load}\ 1; \cdots, [\#\mathsf{I}]\rangle \\ & \to_{\mathsf{h}} & \mathsf{stuck}(\langle \#\mathsf{K} \cdot \#\mathsf{I}, \#\mathsf{I}, \varnothing, \mathtt{Load}\ 2; \cdots, []\rangle) \end{array}$$

Similarly,

$$\mathsf{I} = \mathsf{S} \mathbin{@} [\#\mathsf{K}, \#\mathsf{K}] \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}(\langle \#\mathsf{K}, \#\mathsf{K}, \varnothing, \texttt{Load}\ 2; \cdots, [] \rangle).$$

These two machines are both stuck and different modulo $=_{\mathbb{A}}$ since, e.g., the contents of their register R_1 are $\#\mathsf{I}$ and $\#\mathsf{K}$ respectively, and it is easy to check that $\#\mathsf{I} \not\simeq_{\mathbb{A}} \#\mathsf{K}$. By Theorem 3.11, we conclude that $\#\mathsf{S} \cdot (\#\mathsf{K} \cdot \#\mathsf{I}) \cdot \#\mathsf{I} \not\simeq_{\mathbb{A}} \#\mathsf{I}$. □

4. Lambda Models via Applicative Equivalences

In the previous section we have seen that the equivalence $\simeq_{\mathbb{A}}$, thus $\equiv_{\mathbb{A}}$, is too weak to give rise to a model of λ-calculus (Lemma 3.13). The main problem is that a λ-term $\lambda x.M$ is represented as an addressing machine performing a "`Load`" (to read x from the tape) before evaluating the addressing machine corresponding to M. Since nothing is applied, the tape is empty and the machine gets stuck thus preventing the evaluation of the subterm M. In order to construct a λ-model we introduce the equivalence $\simeq_{\mathbb{A}}^{\text{æ}}$ below.

Definition 4.1. Define the relation $\equiv_{\mathbb{A}}^{\text{æ}}$ as the least equivalence satisfying:

$$\frac{\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =_{\mathbb{A}}^{\text{æ}} \mathsf{N}}{\mathsf{M} \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{N}}\ (\twoheadrightarrow_{\mathbb{A}}^{\text{æ}})$$

$$\frac{\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}(\mathsf{M}') \quad \mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}(\mathsf{N}') \quad \forall a \in \mathbb{A} \,.\, \mathsf{M} \mathbin{@} [a] \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{N} \mathbin{@} [a]}{\mathsf{M} \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{N}}\ (\text{æ})$$

We say that M and N are *applicatively equivalent* whenever $\mathsf{M} \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{N}$. Recall that $\simeq_{\mathbb{A}}^{\text{æ}}$ and $=_{\mathbb{A}}^{\text{æ}}$ are defined in terms of $\equiv_{\mathbb{A}}^{\text{æ}}$ as described in Definition 3.1. Also in this case, it is easy to check that $=_{\mathbb{A}}^{\text{æ}} \subseteq \equiv_{\mathbb{A}}^{\text{æ}}$ holds.

Remark 4.2. The rule (æ) shares similarities with the (ω)-rule in λ-calculus [Bar84, Def. 4.1.10], although being more restricted as only applicable to addressing machine that eventually become stuck. In particular, both rules have countably many premises, therefore a derivation of $\mathsf{M} \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{N}$ is a well-founded ω-branching tree (in particular, the tree is countable and there are no infinite paths). Techniques for performing induction "on the length of a derivation" in this kind of systems are well-established, see e.g. [Bar71, IS06]. More details about the underlying ordinals will be given in Section 5.

Examples 4.3. Convince yourself of the following facts.

(i) As seen in the proof of Lemma 3.13, I and $\mathsf{S} \mathbin{@} [\#\mathsf{K} \cdot \#\mathsf{I}, \#\mathsf{I}]$ both reduce to stuck machines. For all $a \in \mathbb{A}$, we have that

$$\mathsf{I} \mathbin{@} [a] \twoheadrightarrow_{\mathsf{h}} \#^{-1}(a) \;{}_{\mathsf{h}}\!\twoheadleftarrow \mathsf{S} \mathbin{@} [\#\mathsf{K} \cdot \#\mathsf{I}, \#\mathsf{I}, a]\,.$$

By (æ), they are applicatively equivalent.

(ii) Since indeterminate machines x_k are not stuck, $\mathsf{x}_m \equiv_{\mathbb{A}}^{\text{æ}} \mathsf{x}_n$ entails $m = n$.

(iii) Let

$$\mathsf{1} = \langle \varnothing^2, \texttt{Load}\ (0,1); 0 \leftarrow \texttt{App}(0,\,1); \texttt{Call}\ 0, [] \rangle.$$

It is easy to check that, for all $a, b \in \mathbb{A}$, we have $\mathsf{1} \mathbin{@} [a,b] \twoheadrightarrow_{\mathsf{h}} \#^{-1}(a) \mathbin{@} [b] \;{}_{\mathsf{h}}\!\twoheadleftarrow \mathsf{I} \mathbin{@} [a,b]$. However, since $\mathsf{I} \mathbin{@} [\#\mathsf{x}_n] \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ and $\neg\mathsf{stuck}(\mathsf{x}_n)$, one cannot apply (æ), whence (intuitively) they are not applicatively equivalent: $\mathsf{I} \not\equiv_{\mathbb{A}}^{\text{æ}} \mathsf{1}$.

Actually the inequalities claimed in examples (ii)-(iii) above, i.e. $\mathsf{x}_m \not\equiv_{\mathbb{A}}^{\text{æ}} \mathsf{x}_n$ for $m \neq n$ and $\mathsf{I} \not\equiv_{\mathbb{A}}^{\text{æ}} \mathsf{1}$, are difficult to prove formally (see Lemma 4.5(ii)).

Lemma 4.4. *Let* $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$ *and* $a, b \in \mathbb{A}$.

(i) *If* $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ *then* $\mathsf{M} @ [a] \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} @ [a]$.
(ii) *The following rule is derivable:*

$$\frac{\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} \quad a \simeq^{\text{æ}}_{\mathbb{A}} b}{\mathsf{M} @ [a] \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} @ [b]} \text{ (cong)}$$

(iii) *Therefore,* $\simeq^{\text{æ}}_{\mathbb{A}}$ *is a congruence on* $\mathcal{A} = (\mathbb{A}, \cdot, \#\mathsf{K}, \#\mathsf{S})$.

Proof. (i) By induction on a proof of $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$. Possible cases are:

Case ($\twoheadrightarrow^{\text{æ}}_{\mathbb{A}}$). If $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ then $\mathsf{M} @ [a] \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} @ [a] =^{\text{æ}}_{\mathbb{A}} \mathsf{N} @ [a]$, by Lemma 2.12(ii) and the definition of $=^{\text{æ}}_{\mathbb{A}}$.

Case (æ). Trivial, as the thesis is a premise of this rule.

(Symmetry) and (Transitivity) follow from the induction hypothesis.

(ii) Assume that $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ and $a \simeq^{\text{æ}}_{\mathbb{A}} b$. Then, we have:

$$\begin{array}{rcll} \mathsf{M} @ [a] & =^{\text{æ}}_{\mathbb{A}} & \mathsf{M} @ [b], & \text{by reflexivity and } a \simeq^{\text{æ}}_{\mathbb{A}} b, \\ & \equiv^{\text{æ}}_{\mathbb{A}} & \mathsf{N} @ [b], & \text{by } (i). \end{array}$$

So we conclude by transitivity.

(iii) By Lemma 3.2 $\simeq_{\mathbb{A}}$ is an equivalence, by (ii) a congruence. □

We need to show that the congruence $\simeq^{\text{æ}}_{\mathbb{A}}$ is non-trivial, and that the addresses of $\#\mathsf{K}, \#\mathsf{S}$ remain distinguished modulo $\simeq^{\text{æ}}_{\mathbb{A}}$.

Lemma 4.5. *Let* $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$.

(i) *If* $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$ *then* $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$.
(ii) *If* $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ *and* $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ *then* $\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$.
(iii) *Hence, the equivalence relation* $\simeq^{\text{æ}}_{\mathbb{A}}$ *is non-trivial.*
(iv) *In particular,* $\#\mathsf{K} \not\simeq^{\text{æ}}_{\mathbb{A}} \#\mathsf{S}$.

Proof. (i) Easy.

(ii) This proof is the topic of Section 5.

(iii) By (i), the relation is non-empty. By (ii), $\mathsf{x}_i \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{x}_j$ if and only if $i = j$, whence there are infinitely many distinguished equivalence classes.

(iv) From Example 2.13, we get:

$$\begin{array}{lcl} \mathsf{K} @ [\#\mathsf{K}, \#\mathsf{K}, \#\mathsf{x}_1] & \twoheadrightarrow_{\mathsf{h}} & \langle \#\mathsf{x}_1, \texttt{Load } -; \texttt{Call } 0, [] \rangle; \\ \mathsf{S} @ [\#\mathsf{K}, \#\mathsf{K}, \#\mathsf{x}_1] & \twoheadrightarrow_{\mathsf{h}} & \mathsf{x}_1. \end{array}$$

For these machines to be $\equiv^{\text{æ}}_{\mathbb{A}}$-equivalent, the former machine should reduce to x_1, by (ii), which is impossible since $\langle \#\mathsf{x}_1, \texttt{Load } -; \texttt{Call } 0, [] \rangle$ is stuck. □

4.1. **Constructing a λ-model.** We define an interpretation transforming a λ-term with free variables $x_1, \ldots, x_n$ into an addressing machine reading the values of $\vec{x}$ from its tape. The definition is inspired from the well-known categorical interpretation of λ-calculus into a reflexive object of a cartesian closed category. In particular, variables are interpreted as projections. See, e.g., [Koy82] or [Sel02] for more details.

Definition 4.6 (Auxiliary interpretation)**.** Let $M \in \Lambda(\mathbb{A})$ and $x_1, \dots, x_n$ be such that $\mathrm{FV}(M) \subseteq \vec{x}$. Define $|-|_{\vec{x}} : \Lambda(\mathbb{A}) \to \mathcal{M}_{\mathbb{A}}$ by induction as follows:

$$
\begin{array}{lcll}
|x_i|_{\vec{x}} & = & \mathsf{Pr}^n_i, \text{ where } 1 \leq i \leq n; & \\
|\hat{a}|_{\vec{x}} & = & \mathsf{Cons}^n_a, \text{ for } a \in \mathbb{A}; & \\
|MN|_{\vec{x}} & = & \langle \varnothing^n, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}, \varnothing, \mathtt{Apply}_n, [] \rangle; & \\
|\lambda y.M|_{\vec{x}} & = & |M|_{\vec{x},y}, & \text{assuming wlog that } y \notin \vec{x};
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathsf{Pr}^n_i & = & \langle \varnothing, (\mathtt{Load}\ -)^{i-1}; \mathtt{Load}\ 0; (\mathtt{Load}\ -)^{n-i-1}; \mathtt{Call}\ 0, [] \rangle, \\
\mathsf{Cons}^n_a & = & \langle a, (\mathtt{Load}\ -)^n; \mathtt{Call}\ 0, [] \rangle, \\
\mathtt{Apply}_n & = & \mathtt{Load}\ (0, \dots, n-1); n \leftarrow \mathtt{App}(n,\, 0); \cdots ; n \leftarrow \mathtt{App}(n,\, n-1); \\
& & n+1 \leftarrow \mathtt{App}(n+1,\, 0); \cdots ; n+1 \leftarrow \mathtt{App}(n+1,\, n-1); \\
& & n+2 \leftarrow \mathtt{App}(n,\, n+1); \mathtt{Call}\ n+2.
\end{array}
$$

Remark 4.7. Let $n \in \mathbb{N}$, and $T = [a_1, \dots, a_n] \in \mathbb{T}_{\mathbb{A}}$. We have:

(i) $\mathsf{Pr}^n_i \,@\, T \twoheadrightarrow_{\mathsf{h}} \#^{-1}(a_i)$, for all $i\,(1 \leq i \leq n)$;
(ii) $\mathsf{Cons}^n_b \,@\, T \twoheadrightarrow_{\mathsf{h}} \#^{-1}(b)$, for all $b \in \mathbb{A}$;
(iii) $\langle \varnothing^n, \#\mathsf{M}, \#\mathsf{N}, \varnothing, \mathtt{Apply}_n, T \rangle \twoheadrightarrow_{\mathsf{h}} (\mathsf{M} \,@\, T\,) \,@\, [\#(\mathsf{N} \,@\, T\,)]\,.$

From now on, whenever writing $|M|_{\vec{x}}$, we assume that $\mathrm{FV}(M) \subseteq \vec{x}$. The following are basic properties of the interpretation map defined above.

Lemma 4.8. *Let $M \in \Lambda(\mathbb{A})$, $n \in \mathbb{N}$, $\vec{x} = x_1, \dots, x_n$ and $\vec{a} = a_1, \dots, a_n \in \mathbb{A}$.*

(i) $|M|_{\vec{x}} = \langle \vec{R}, \mathtt{Load}\ (i_1, \dots, i_n); P, [] \rangle$ *for some $\mathbb{A}_\varnothing$-valued registers $\vec{R}$, program P and indices $i_j \in \mathbb{N}$.*
(ii) *If $m < n$ then $|M|_{\vec{x}} \,@\, [a_1, \dots, a_m] \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}()$.*
(iii) *For all $b \in \mathbb{A}$, we have $|M|_{y,\vec{x}} \,@\, [b] \equiv^{\text{æ}}_{\mathbb{A}} |M[\hat{b}/y]|_{\vec{x}}$.*
(iv) *In particular, if $y \notin \mathrm{FV}(M)$ then $|M|_{y,\vec{x}} \,@\, [b] \equiv^{\text{æ}}_{\mathbb{A}} |M|_{\vec{x}}$.*
(v) $|M|_{\vec{x}} \,@\, [\vec{a}\,] \equiv^{\text{æ}}_{\mathbb{A}} |M|_{x_{\sigma(1)},\dots,x_{\sigma(n)}} \,@\, [a_{\sigma(1)}, \dots, a_{\sigma(n)}]$ *for all permutations σ of $\{1, \dots, n\}$.*

Proof of Lemma 4.8. (i) By a straightforward induction on M.

(ii) It follows from (i).

(iii) We proceed by structural induction on M. By (ii), if $\vec{x} \neq \emptyset$ then both addressing machines reduce to stuck ones, so we can test the applicative equivalence by applying an arbitrary $\vec{a}$ and conclude using (æ) n-times.

Case $M = \hat{c}$. Then $c[\hat{b}/y] = c$, and we have:

$$|\hat{c}|_{y,\vec{x}} \,@\, [b, \vec{a}] = \mathsf{Cons}^{n+1}_c \,@\, [b, \vec{a}] \twoheadrightarrow_{\mathsf{h}} \#^{-1}(c) \,{}_{\mathsf{h}}\!\twoheadleftarrow \mathsf{Cons}^n_c \,@\, [\vec{a}]\,.$$

Case $M = x_i$ for some $i\,(1 \leq i \leq n)$. Then $x_i[\hat{b}/y] = x_i$ and

$$|x_i|_{y,\vec{x}} \,@\, [b, \vec{a}] = \mathsf{Pr}^{n+1}_{i+1} \,@\, [b, \vec{a}] \twoheadrightarrow_{\mathsf{h}} \#^{-1}(a_i) \,{}_{\mathsf{h}}\!\twoheadleftarrow \mathsf{Pr}^n_i \,@\, [\vec{a}] = |x_i|_{\vec{x}} \,@\, [\vec{a}]\,.$$

Case $M = y$. Then $y[\hat{b}/y] = \hat{b}$ and we have:

$$|y|_{y,\vec{x}} \,@\, [b, \vec{a}\,] = \mathsf{Pr}^{n+1}_1 \,@\, [b, \vec{a}\,] \twoheadrightarrow_{\mathsf{h}} \#^{-1}(b) \,{}_{\mathsf{h}}\!\twoheadleftarrow \mathsf{Cons}^n_b \,@\, [\vec{a}\,] = |\hat{b}|_{\vec{x}} \,@\, [\vec{a}\,]\,.$$

Case $M = PQ$. Then $(PQ)[\hat{b}/y] = (P[\hat{b}/y])(Q[\hat{b}/y])$ and we have:

$$\begin{array}{rcl} |PQ|_{y,\vec{x}} @ [b,\vec{a}\,] & = & \langle \varnothing^{n+1}, \#|P|_{y,\vec{x}}, \#|Q|_{y,\vec{x}}, \varnothing, \mathtt{Apply}_{n+1}, [b,\vec{a}\,]\rangle \\ & \twoheadrightarrow_{\mathsf{h}} & |P|_{y,\vec{x}} @ [b,\vec{a}, \#(|Q|_{y,\vec{x}} @ [b,\vec{a}\,]\,)] \\ & \equiv^{\text{æ}}_{\mathbb{A}} & |P[\hat{b}/y]|_{\vec{x}} @ [\vec{a}, \#(|Q[\hat{b}/y]|_{\vec{x}} @ [\vec{a}\,]\,)]\,,\ \text{by IH}, \\ & {}_{\mathsf{c}}\twoheadleftarrow & \langle \varnothing^{n}, \#|P[\hat{b}/y]|_{\vec{x}}, \#|Q[\hat{b}/y]|_{\vec{x}}, \varnothing, \mathtt{Apply}_{n}, [\vec{a}]\rangle \\ & = & |(P[\hat{b}/y])(Q[\hat{b}/y])|_{\vec{x}} \\ & = & |(PQ)[\hat{b}/y]|_{\vec{x}} \end{array}$$

Case $M = \lambda z.P$, wlog $z \notin y, \vec{x}$, so $(\lambda z.P)[\hat{b}/y] = \lambda z.P[\hat{b}/y]$. By (ii) both machines reduce to stuck ones. So we have to apply an extra $a_{n+1} \in \mathbb{A}$.

$$\begin{array}{rcll} |\lambda z.P|_{y,\vec{x}} @ [b,\vec{a},a_{n+1}] & = & |P|_{y,\vec{x},z} @ [b,\vec{a},a_{n+1}] & \\ & \equiv^{\text{æ}}_{\mathbb{A}} & |P[\hat{b}/x]|_{\vec{x},z} @ [\vec{a},a_{n+1}]\,, & \text{by IH}, \\ & = & |\lambda z.P[\hat{b}/y]|_{\vec{x}} @ [\vec{a},a_{n+1}] & \end{array}$$

(iv) By (iii).

(v) By (iv), permuting the substitutions. □

Definition 4.9. Let $\mathcal{S} = (\mathbb{A}/_{\simeq^{\text{æ}}_{\mathbb{A}}}, \bullet, [\![-]\!]_{-})$, where

$$\begin{array}{rcl} [a]_{\simeq^{\text{æ}}_{\mathbb{A}}} \bullet [b]_{\simeq^{\text{æ}}_{\mathbb{A}}} & = & [a \cdot b]_{\simeq^{\text{æ}}_{\mathbb{A}}}, \\ [\![M]\!]_{\rho} & \simeq^{\text{æ}}_{\mathbb{A}} & \#(|M|_{\vec{x}} @ [\rho(x_1),\dots,\rho(x_n)]\,). \end{array}$$

By Lemma 4.8, the definition of $[\![M]\!]_{\rho}$ is independent from the choice of $\vec{x}$, as long as $\mathrm{FV}(M) \subseteq \vec{x}$. This is reminiscent of the standard way for defining a syntactic interpretation from a categorical one. (Again, see Koymans's [Koy82].)

Theorem 4.10. *$\mathcal{S}$ is a syntactic λ-model.*

Proof. We need to check that the conditions (i)–(vi) from Definition 1.9 are satisfied by the interpretation function given in Definition 4.9.

Take $\vec{x} = x_1,\dots,x_n$, and write $\rho(\vec{x})$ for $\rho(x_1),\dots,\rho(x_n)$.

(i) $[\![x_i]\!]_{\rho} \simeq^{\text{æ}}_{\mathbb{A}} \#(\mathsf{Pr}^n_i @ [\rho(\vec{x})]\,) \simeq^{\text{æ}}_{\mathbb{A}} \rho(x_i)$, by Remark 4.7(i).

(ii) $[\![\hat{a}]\!]_{\rho} \simeq^{\text{æ}}_{\mathbb{A}} \#(\mathsf{Cons}^n_a @ [\rho(\vec{x})]\,) \simeq^{\text{æ}}_{\mathbb{A}} a$, by Remark 4.7(ii).

(iii) In the application case, we have:

$$\begin{array}{rcll} [\![PQ]\!]_{\rho} & \simeq^{\text{æ}}_{\mathbb{A}} & \#(|PQ|_{\vec{x}} @ [\rho(\vec{x})]\,) & \\ & = & \langle \varnothing^n, \#|P|_{\vec{x}}, \#|Q|_{\vec{x}}, \varnothing, \mathtt{Apply}_n, [\rho(\vec{x})]\rangle, & \text{by Def. 4.6}, \\ & \simeq^{\text{æ}}_{\mathbb{A}} & \#(\mathsf{M} @ [\rho(\vec{x})]\,) \cdot \#(\mathsf{N} @ [\rho(\vec{x})]\,), & \text{by Rem. 4.7(iii)}, \\ & = & [\![P]\!]_{\rho} \bullet [\![Q]\!]_{\rho} & \end{array}$$

(iv) In the λ-abstraction case we have, for all $a \in \mathbb{A}$:

$$[\![\lambda y.P]\!]_{\rho} \cdot a \simeq^{\text{æ}}_{\mathbb{A}} |\lambda y.P|_{\vec{x}} @ [\rho(\vec{x}), a] \simeq^{\text{æ}}_{\mathbb{A}} |P|_{\vec{x},y} @ [\rho(\vec{x}), a] \simeq^{\text{æ}}_{\mathbb{A}} [\![P]\!]_{\rho[y:=a]}.$$

(v) This follows from Lemma 4.8(iv).

(vi) By definition $[\![\lambda y.M]\!]_{\rho} \simeq^{\text{æ}}_{\mathbb{A}} \#(|M|_{\vec{x},y} @ [\rho(\vec{x})]\,)$ and, by Lemma 4.8(ii), $|M|_{\vec{x},y} @ [\rho(\vec{x})]$ reduces to a stuck addressing machine. Similarly, for $[\![\lambda x.N]\!]_{\rho}$. We conclude by applying the rule (æ). □

Remark 4.11. (i) For closed λ-terms $M \in \Lambda^o$, we have $[\![M]\!] = |M|$.

(ii) It is easy to check that $[\![\mathbf{K}]\!] \simeq^{\text{æ}}_{\mathbb{A}} \#\mathsf{K}$ and $[\![\mathbf{S}]\!] \simeq^{\text{æ}}_{\mathbb{A}} \#\mathsf{S}$.

(iii) More generally, all addressing machines behaving as the combinator $\mathbf{K}$ (resp. $\mathbf{S}$) are equated in the model.

Lemma 4.12. *The syntactic λ-model $\mathcal{S}$ is not extensional.*

Proof. It is enough to check that $\mathcal{S} \not\models \mathbf{1} = \mathbf{I}$. Now, we have:

$$\begin{array}{rcl} [\![\, \mathbf{1} \,]\!] & = & \langle \varnothing^2, \#\mathsf{Pr}^2_1, \#\mathsf{Pr}^2_2, \varnothing, \mathtt{Apply}_2, [] \rangle; \\ [\![\, \mathbf{I} \,]\!] & = & \langle \varnothing, \mathtt{Load}\ 0; \mathtt{Call}\ 0, [] \rangle. \end{array}$$

By applying an indeterminate machine x_n, the former reduces to a stuck machine, while the latter reduces to x_n. By Lemma 4.5(ii), they must be different modulo $\equiv^{æ}_{\mathbb{A}}$. □

A difficult problem that arises naturally is the characterization of the λ-theory induced by the λ-model $\mathcal{S}$ defined above.

Proposition 4.13. *The λ-theory* $\mathrm{Th}(\mathcal{S})$ *is neither extensional nor sensible.*

Proof. $\mathrm{Th}(\mathcal{S})$ is not extensional by Lemma 4.12. To show that it is not sensible, it is enough to check that $\mathcal{S} \not\models \lambda x.\mathbf{\Omega} = \mathbf{\Omega}$. Notice that

$$\begin{array}{rcll} |\mathbf{\Omega}| & = & \langle \#|\mathbf{\Delta}|, \#|\mathbf{\Delta}|, \varnothing, 2 \leftarrow \mathtt{App}(0,\, 1); \mathtt{Call}\ 2, [] \rangle, & \\ & \to_{\mathsf{h}} & |\mathbf{\Delta}| \mathbin{@} [\#|\mathbf{\Delta}|]\,, & \text{where:} \\ |\mathbf{\Delta}| & = & \langle \varnothing, \#\mathsf{Pr}^1_1, \#\mathsf{Pr}^1_1, \varnothing, \mathtt{Apply}_1, [] \rangle. & \end{array}$$

By induction on a derivation of $\mathsf{M} \equiv^{æ}_{\mathbb{A}} \mathsf{N}$, one checks that $\mathsf{M} \equiv^{æ}_{\mathbb{A}} \mathsf{N}$ and $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{D}_1 \mathbin{@} [\mathsf{D}_2]$ with $\mathsf{D}_1 \simeq^{æ}_{\mathbb{A}} \mathsf{D}_2 \simeq^{æ}_{\mathbb{A}} |\mathbf{\Delta}|$entails $\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{D}'_1 \mathbin{@} [\#\mathsf{D}'_2]$ for some $\mathsf{D}'_1 \simeq^{æ}_{\mathbb{A}} \mathsf{D}'_2 \simeq^{æ}_{\mathbb{A}} |\mathbf{\Delta}|$. We conclude because the machine $|\lambda x.\mathbf{\Omega}|$ is stuck. □

5. Consistency Proof via Ordinal Analysis

In this section we adapt Barendregt's proof of consistency of $\boldsymbol{\lambda}\omega$ (the least λ-theory closed under the (ω)-rule) to prove Lemma 4.5(ii), which entails the consistency of our system. First, we need to introduce in our setting the notion of *context* and *underlined reduction*, that are omnipresent techniques in the area of term rewriting systems.

5.1. Contexts and Underlined Head Reductions.

In λ-calculus a context is a λ-term possibly containing occurrences of an algebraic variable, called *hole*, that can be substituted by any λ-term possibly with capture of free variables. We will define a *context-machine* similarly, namely as an addressing machine possibly having a "hole" denoted by ξ. Formally, we introduce a new machine having no registers or program, only an empty tape (therefore distinguished from all machines populating $\mathcal{M}_{\mathbb{A}}$):

$$\xi = \langle [] \rangle$$

We then extend our formalism to include machines working either directly or indirectly with one, or more, occurrences of ξ. We wish to ensure the invariant that a machine M with no occurrences of ξ maintain as address $\#\mathsf{M}$ — for this reason we need to extend the range of addresses in a conservative way.

Consider a countable set $\mathbb{B}$ of addresses such that $\mathbb{A} \cap \mathbb{B} = \emptyset$, and write $\mathbb{X} = \mathbb{A} \cup \mathbb{B}$ for the set of *extended addresses*. As usual, we set

$$\mathbb{X}_{\varnothing} = \mathbb{X} \cup \{\varnothing\}.$$

Definition 5.1. (i) An *extended machine* X is either of the form
- $\xi @ T$ or
- $\langle \vec{R}, P, T\rangle$

where $\vec{R}$ are $\mathbb{X}_{\varnothing}$-valued registers, P is a valid program, $T \in \mathbb{T}_{\mathbb{X}}$ is an $\mathbb{X}$-valued tape. We write $\mathcal{M}^{\xi}_{\mathbb{X}}$ for the set of all extended machines.

(ii) Fix a bijective map $\underline{\#} : \mathcal{M}^{\xi}_{\mathbb{X}} \to \mathbb{X}$ satisfying $\underline{\#}(\mathsf{M}) = \#\mathsf{M}$ for all addressing machine $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$. Write $\underline{\#}^{-1}(\cdot) : \mathbb{X} \to \mathcal{M}^{\xi}_{\mathbb{X}}$ for its inverse.

(iii) The *number of occurrences* of ξ in $\mathsf{X} \in \mathcal{M}^{\xi}_{\mathbb{X}}$ (resp. R_i, resp. T), written $\mathrm{occ}_{\xi}(\mathsf{X}) \in \mathbb{N} \cup \{\infty\}$ ($\mathrm{occ}_{\xi}(R_i)$, $\mathrm{occ}_{\xi}(T) \in \mathbb{N} \cup \{\infty\}$), is defined as follows:

$$\begin{array}{lcl}
\mathrm{occ}_{\xi}(\xi @ T) & = & 1 + \mathrm{occ}_{\xi}(T);\\
\mathrm{occ}_{\xi}(\langle \vec{R}, P, T\rangle) & = & \mathrm{occ}_{\xi}(T) + \sum_{i=0}^{r-1} \mathrm{occ}_{\xi}(R_i);\\
\mathrm{occ}_{\xi}([a_1, \dots, a_n]) & = & \mathrm{occ}_{\xi}(\underline{\#}^{-1}(a_1)) + \cdots + \mathrm{occ}_{\xi}(\underline{\#}^{-1}(a_n));\\
\mathrm{occ}_{\xi}(R_i) & = & \begin{cases} 0, & \text{if } R_i = \varnothing,\\ \mathrm{occ}_{\xi}(\underline{\#}^{-1}(a)), & \text{if } R_i = a \in \mathbb{X}. \end{cases}
\end{array}$$

Notice that $\mathrm{occ}_{\xi}(\mathsf{M}) \in \mathbb{N}$ entails that $\mathrm{occ}_{\xi}(\mathsf{M}.R_i), \mathrm{occ}_{\xi}(\mathsf{M}.T) \in \mathbb{N}$.

The number of occurrences of ξ in an extended machine X has been defined to handle the fact that recursively dereferencing all the addresses contained in an extended addressing machine might result in a non-terminating process (see Remark 2.9).

Examples 5.2. The following are examples of extended machines:

(i) ξ, with $\mathrm{occ}_{\xi}(\xi) = 1$;
(ii) $\mathsf{K} @ [\underline{\#}\xi, \underline{\#}(\xi @ [\underline{\#}\xi])]$, with $\mathrm{occ}_{\xi}(\mathsf{K} @ [\underline{\#}\xi, \underline{\#}(\xi @ [\underline{\#}\xi])]) = 3$;
(iii) for all $n \in \mathbb{N}$, $\mathsf{X}_n = \langle \underline{\#}\xi, \varepsilon, [\underline{\#}\mathsf{X}_{n+1}]\rangle$. In this case, $\mathrm{occ}_{\xi}(\mathsf{X}_0) = \infty$.

As previously mentioned, a key property of contexts in λ-calculus is that one can plug a λ-term into the hole and obtain a regular λ-term. Similarly, given $\mathsf{M} \in \mathcal{M}^{\xi}_{\mathbb{X}}$ and $\mathsf{X} \in \mathcal{M}^{\xi}_{\mathbb{X}}$, we can define the addressing machine $\mathsf{X}(\!|\mathsf{M}|\!)$ obtained from X by recursively substituting (even in the registers/tapes) each occurrence of ξ by M. However, this operation is well-defined only when $\mathrm{occ}_{\xi}(\mathsf{X})$ is finite, so we focus on extended machines enjoying this property.

Definition 5.3.

(i) A *context-machine* is any $\mathsf{C} \in \mathcal{M}^{\xi}_{\mathbb{X}}$ satisfying $\mathrm{occ}_{\xi}(\mathsf{C}) \in \mathbb{N}$.
(ii) Given a context-machine C and $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$, define the addressing machine $\mathsf{C}(\!|\mathsf{M}|\!)$ as follows:

$$\mathsf{C}(\!|\mathsf{M}|\!) = \begin{cases} \mathsf{M} @ T(\!|\mathsf{M}|\!), & \text{if } \mathsf{C} = \xi @ T,\\ \langle \vec{R}(\!|\mathsf{M}|\!), P, T(\!|\mathsf{M}|\!)\rangle, & \text{if } \mathsf{C} = \langle \vec{R}, P, T\rangle; \end{cases}$$

where (assuming $a \in \mathbb{X}, T = [a_1, \dots, a_n] \in \mathbb{T}_{\mathbb{X}}$ with $\mathrm{occ}_{\xi}(a :: T) \in \mathbb{N}$):

$$\begin{array}{lcl}
a(\!|\mathsf{M}|\!) & = & \underline{\#}(\underline{\#}^{-1}(a)(\!|\mathsf{M}|\!));\\
R_i(\!|\mathsf{M}|\!) & = & \begin{cases} \varnothing & \text{if } R_i = \varnothing,\\ a(\!|\mathsf{M}|\!) & \text{if } R_i = a; \end{cases}\\
T(\!|\mathsf{M}|\!) & = & [a_1(\!|\mathsf{M}|\!), \dots, a_n(\!|\mathsf{M}|\!)].
\end{array}$$

In the following, when writing $\mathsf{C}(\!|\mathsf{M}|\!)$ (resp. $a(\!|\mathsf{M}|\!)$, $R_i(\!|\mathsf{M}|\!)$, $T(\!|\mathsf{M}|\!)$) we silently assume that the number of occurrences of ξ in C (resp. a, R_i, T) is finite. Let us introduce a notion of reduction for context-machines that allows to mimic the underlined reduction from [Bar71]. The idea is to decompose a machine N as $\mathsf{N} = \mathsf{C}(\!|\underline{\mathsf{M}}|\!)$ where C is a context-machine and M the underlined sub-machine. It is now possible to reduce C independently from M until either the machine reaches a final-state or ξ reaches the head-position. In the latter case, we substitute the head occurrence of ξ by M, and continue the computation.

Definition 5.4.

(i) The head reduction $\to_{\mathsf{h}}$ is generalized to extended machines in the obvious way, using $\underline{\#}(\cdot)$ rather than $\#(\cdot)$ to compute the addresses. In particular, the machine $\xi \mathbin{@} T \not\to_{\mathsf{h}}$ is in final state, but it is not stuck.

(ii) Given $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ and $\mathsf{C} \in \mathcal{M}^{\xi}_{\mathbb{A}}$, the M-*underlined (head-)reduction* $\to^{\mathsf{M}}_{\mathsf{h}}$ is defined by adding to (i) the rule

$$\xi \mathbin{@} T \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{M} \mathbin{@} T\,.$$

Examples 5.5. Let $\mathsf{C} = \mathsf{S} \mathbin{@} [\underline{\#}\xi, \underline{\#}\xi, \underline{\#}\mathsf{x}_n]$. Then $\mathsf{C}(\!|\mathsf{K}|\!) = \mathsf{S} \mathbin{@} [\#\mathsf{K}, \#\mathsf{K}, \#\mathsf{x}_n]$.

(i) $\mathsf{C} \twoheadrightarrow_{\mathsf{h}} \xi \mathbin{@} [\underline{\#}\mathsf{x}_n, \underline{\#}(\xi \mathbin{@} [\underline{\#}\mathsf{x}_n])]$.

(ii) $\mathsf{C} \twoheadrightarrow^{\mathsf{K}}_{\mathsf{h}} \xi \mathbin{@} [\underline{\#}\mathsf{x}_n, \underline{\#}(\xi \mathbin{@} [\underline{\#}\mathsf{x}_n])] \to^{\mathsf{K}}_{\mathsf{h}} \mathsf{K} \mathbin{@} [\underline{\#}\mathsf{x}_n, \underline{\#}(\xi \mathbin{@} [\underline{\#}\mathsf{x}_n])] \twoheadrightarrow^{\mathsf{K}}_{\mathsf{h}} \mathsf{x}_n$.

Lemma 5.6. *For $\mathsf{C}, \mathsf{C}' \in \mathcal{M}^{\xi}_{\mathbb{X}}$ and $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$, the following are equivalent:*

(1) $\mathsf{C}(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$;

(2) $\mathsf{C} \twoheadrightarrow^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}'$ *and* $\mathsf{C}'(\!|\mathsf{M}|\!) = \mathsf{N}$.

Proof. ($1 \Rightarrow 2$) By induction on the length n of the reduction $\mathsf{C}(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$.

Case $n = 0$. Trivial, take $\mathsf{C}' = \mathsf{C}$.

Case $n > 0$. Let $\mathsf{C}(\!|\mathsf{M}|\!) \to_{\mathsf{h}} \mathsf{N}' \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$. Split into cases depending on C.

Subcase $\mathsf{C} = \xi \mathbin{@} T$, therefore $\mathsf{C}(\!|\mathsf{M}|\!) = \mathsf{M} \mathbin{@} T(\!|\mathsf{M}|\!) \to_{\mathsf{h}} \mathsf{N}'$. There are two possibilities:

- M is stuck and $T \neq []$, say, $T = [a_0, \dots, a_n]$. In this case $\mathsf{C}(\!|\mathsf{M}|\!) = \langle \vec{R}, \texttt{Load}\ i; P, [] \rangle$ and $\mathsf{N}' = \langle \vec{R}[R_i := a_0(\!|\mathsf{M}|\!)], \texttt{Load}\ i; P, [a_1(\!|\mathsf{M}|\!), \dots, a_n(\!|\mathsf{M}|\!)] \rangle$. On the other side, $\mathsf{C} \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{M} \mathbin{@} T \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}''$ for
$$\mathsf{C}'' = \langle \vec{R}[R_i := a_0], \texttt{Load}\ i; P, [a_1, \dots, a_n] \rangle$$
satisfying $\mathsf{C}''(\!|\mathsf{M}|\!) = \mathsf{N}' \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$. We conclude by induction hypothesis.
- $\mathsf{M} \to_{\mathsf{h}} \mathsf{M}'$. In this case $\mathsf{N}' = \mathsf{M}' \mathbin{@} T(\!|\mathsf{M}|\!)$ and $\mathsf{C} \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{M} \mathbin{@} T \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}''$ for $\mathsf{C}'' = \mathsf{M}' \mathbin{@} T$ satisfying $\mathsf{C}''(\!|\mathsf{M}|\!) = \mathsf{N}' \twoheadrightarrow_{\mathsf{h}} \mathsf{N}$. We conclude by induction hypothesis.

Subcase $\mathsf{C} = \langle \vec{R}, P, T \rangle$. By case analysis on P. All cases follow easily from the induction hypothesis.

($2 \Rightarrow 1$) By induction on the length n of the reduction $\mathsf{C} \twoheadrightarrow^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}'$.

Case $n = 0$. Trivial, take $\mathsf{N} = \mathsf{C}(\!|\mathsf{M}|\!)$.

Case $n > 0$, i.e. $\mathsf{C} \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}'' \twoheadrightarrow^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}'$, where the latter reduction is shorter.

Proceed by case analysis on the shape of C.

Subcase $\mathsf{C} = \xi \mathbin{@} T$ and $\mathsf{C}'' = \mathsf{M} \mathbin{@} T$. Then $\mathsf{N} = \mathsf{C}''(\!|\mathsf{M}|\!) = \mathsf{M} \mathbin{@} T(\!|\mathsf{M}|\!) = \mathsf{C}(\!|\mathsf{M}|\!)$. Conclude by induction hypothesis.

Subcase $\mathsf{C} = \langle \vec{R}, P, T \rangle$. By case analysis on P. All cases follow easily from the induction hypothesis. □

$$\frac{\mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}}{\mathsf{M} \approx_0 \mathsf{N}}\ (\approx_0) \qquad \frac{\mathsf{M} \approx_\alpha \mathsf{N}}{\mathsf{M} \sim_\alpha \mathsf{N}}\ (\subseteq^{\approx}_{\alpha}) \qquad \frac{\mathsf{M} \sim_\alpha \mathsf{N}}{\mathsf{M} \equiv_\alpha \mathsf{N}}\ (\subseteq^{\sim}_{\alpha})$$

$$\frac{\mathsf{M},\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}() \quad \forall a \in \mathbb{A}, \exists \gamma < \alpha \,.\, \mathsf{M} \,@\, [a] \ \equiv_\gamma \mathsf{N} \,@\, [a]}{\mathsf{M} \approx_\alpha \mathsf{N}}\ (\approx_\alpha)$$

$$\frac{\#^{-1}(a) \sim_\alpha \#^{-1}(b)}{\mathsf{M}[R_i := a] \sim_\alpha \mathsf{M}[R_i := b]}\ (R^{\sim}_{\alpha}) \qquad \frac{\#^{-1}(a) \sim_\alpha \#^{-1}(b)}{\mathsf{M} \,@\, [a] \ \sim_\alpha \mathsf{M} \,@\, [b]}\ (@^{\sim}_{\alpha})$$

$$\frac{\mathsf{M} \sim_\alpha \mathsf{N} \quad T \in \mathbb{T}_{\mathbb{A}}}{\mathsf{M} \,@\, T \ \sim_\alpha \mathsf{N} \,@\, T}\ (T^{\sim}_{\alpha}) \qquad \frac{\mathsf{M} \equiv_\alpha \mathsf{N} \quad T \in \mathbb{T}_{\mathbb{A}}}{\mathsf{M} \,@\, T \ \equiv_\alpha \mathsf{N} \,@\, T}\ (T_\alpha)$$

$$\frac{\#^{-1}(a) \equiv_\alpha \#^{-1}(b)}{\mathsf{M}[R_i := a] \equiv_\alpha \mathsf{M}[R_i := b]}\ (R_\alpha) \qquad \frac{\#^{-1}(a) \equiv_\alpha \#^{-1}(b)}{\mathsf{M} \,@\, [a] \ \equiv_\alpha \mathsf{M} \,@\, [b]}\ (@_\alpha)$$

$$\frac{\mathsf{M} \approx_\gamma \mathsf{N} \quad \gamma \leq \alpha}{\mathsf{M} \approx_\alpha \mathsf{N}}\ (\leq^{\approx}_{\alpha}) \quad \frac{\mathsf{M} \sim_\gamma \mathsf{N} \quad \gamma \leq \alpha}{\mathsf{M} \sim_\alpha \mathsf{N}}\ (\leq^{\sim}_{\alpha}) \quad \frac{\mathsf{M} \equiv_\gamma \mathsf{N} \quad \gamma \leq \alpha}{\mathsf{M} \equiv_\alpha \mathsf{N}}\ (\leq_\alpha)$$

$$\frac{\mathsf{M} \equiv_\alpha \mathsf{Z} \quad \mathsf{Z} \equiv_\alpha \mathsf{N}}{\mathsf{M} \equiv_\alpha \mathsf{N}}\ (\mathrm{Tr}_\alpha)$$

Figure 1: Rules satisfied by $\approx_\alpha$, $\sim_\alpha$ and $\equiv_\alpha$, beyond reflexivity and symmetry.

5.2. **Ordinal analysis.** As mentioned in Remark 4.2, a derivation of $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ has the structure of a well-founded ω-branching tree. Unfortunately, this makes it difficult to prove even simple properties like Lemma 4.5(ii). We need a more refined system exposing the underlying ordinal and handling the applications of the (Transitivity) rule separately.

Definition 5.7. (i) Let ω_1 be the set of all countable ordinals.

(ii) If π is a derivation of $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$, we define its *length* $\ell(\pi) \in \omega_1$ in the usual inductive way for the rules $(\twoheadrightarrow^{\text{æ}}_{\mathbb{A}})$, (Refl.), (Symm.), (Trans.). Concerning the rule (æ) having countably many premises, we set:

$$\ell\left(\frac{\mathsf{M},\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}() \quad \forall a \in \mathbb{A} \,.\, \overline{\mathsf{M} \,@\, [a] \ \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} \,@\, [a]}^{\,\pi_a}}{\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}}\right) = \sup_{a \in \mathbb{A}}(\ell(\pi_a) + 1)$$

It is easy to check that, if a derivation π has premises $(\pi_i)_{i \in \mathcal{I}}$ for some countable set $\mathcal{I}$ then $\ell(\pi) > \ell(\pi_i)$ for every $i \in \mathcal{I}$.

(iii) For all $\alpha \in \omega_1$, define $\equiv_\alpha, \sim_\alpha, \approx_\alpha \subseteq \mathcal{M}^2_{\mathbb{A}}$ as the least reflexive and symmetric relations closed under the rules of Figure 1.

The intuitive meanings of the relations $\equiv_\alpha, \sim_\alpha, \approx_\alpha$ are the following:

- $\mathsf{M} \equiv_\alpha \mathsf{N} \iff \mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ is derivable using the rule (æ) at most α times;
- $\mathsf{M} \sim_\alpha \mathsf{N} \iff \mathsf{M} \equiv_\alpha \mathsf{N}$ is derivable without using transitivity;
- $\mathsf{M} \approx_\alpha \mathsf{N} \iff \mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$ in case $\alpha = 0$. Otherwise, if $\alpha > 0$ then
- $\mathsf{M} \approx_\alpha \mathsf{N} \iff \mathsf{M} \sim_\alpha \mathsf{N}$ follows directly from the rule (æ).

More precisely, the rules $(\approx_0)$, $(\subseteq^{\approx}_{\alpha})$, $(\subseteq^{\sim}_{\alpha})$ express the fact that $\equiv_{\mathbb{A}} \subseteq \approx_\alpha \subseteq \sim_\alpha \subseteq \equiv_\alpha$. The rule $(\approx_\alpha)$ allows to prove $\mathsf{M} \approx_\alpha \mathsf{N}$, provided that both machines eventually get stuck and that $\mathsf{M} \,@\, [a] \ \equiv_{\gamma_a} \mathsf{N} \,@\, [a]$ is provable for every address a, using a smaller ordinal $\gamma_a < \alpha$. The rules (R_α), $(@_\alpha)$ and (T_α) (resp. $(R^{\sim}_{\alpha})$, $(@^{\sim}_{\alpha})$ and $(T^{\sim}_{\alpha})$) represent the contextuality of the relation $\equiv_\alpha$ (resp. $\sim_\alpha$). The rules $(\leq^{\approx}_\alpha)$, $(\leq^{\sim}_\alpha)$ and $(\leq_\alpha)$ specify that incrementing the ordinal (from top to bottom) is always allowed. Finally, (Tr_α) gives the transitivity of $\equiv_\alpha$.

The following lemma describes formally the intuitive meaning discussed above.

Lemma 5.8. *Let* $\mathsf{M},\mathsf{N} \in \mathcal{M}_{\mathbb{A}}$

(i) $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} \iff \exists \alpha \in \omega_1 \,.\, \mathsf{M} \equiv_\alpha \mathsf{N}$.
(ii) $\mathsf{M} \equiv_0 \mathsf{N} \iff \mathsf{M} \equiv_{\mathbb{A}} \mathsf{N}$.
(iii) $\mathsf{M} \equiv_\alpha \mathsf{N} \iff \exists n \geq 0, \mathsf{Z}_1,\dots,\mathsf{Z}_n \in \mathcal{M}_{\mathbb{A}} \,.\, \mathsf{M} \sim_\alpha \mathsf{Z}_1 \sim_\alpha \cdots \sim_\alpha \mathsf{Z}_n = \mathsf{N}$.
(iv) $\mathsf{M} \sim_\alpha \mathsf{N} \iff \exists \mathsf{C} \in \mathcal{M}^{\xi}_{\mathbb{X}}, \mathsf{M}',\mathsf{N}' \in \mathcal{M}_{\mathbb{A}} \,.$ $\mathsf{M} = \mathsf{C}(\!|\mathsf{M}'|\!) \land \mathsf{N} = \mathsf{C}(\!|\mathsf{N}'|\!) \land \mathsf{M}' \approx_\alpha \mathsf{N}'$.
(v) $\mathsf{M} \approx_\alpha \mathsf{N} \land \alpha \neq 0 \iff \mathsf{M},\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}() \land$ $\forall a \in \mathbb{A}, \exists \gamma < \alpha \,.\, \mathsf{M} \mathbin{@} [a] \equiv_\gamma \mathsf{N} \mathbin{@} [a]$.

Proof. (i) $(\Leftarrow)$ Easy.

$(\Rightarrow)$ By induction on the length of a derivation of $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N}$.

Case $(\twoheadrightarrow^{\text{æ}}_{\mathbb{A}})$. I.e., there exists $\mathsf{Z} \in \mathcal{M}_{\mathbb{A}}$ such that $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{Z} =^{\text{æ}}_{\mathbb{A}} \mathsf{N}$. By Theorem 3.11, we have $\mathsf{M} \equiv_{\mathbb{A}} \mathsf{Z}$ whence $\mathsf{M} \equiv_0 \mathsf{Z}$ by $(\approx_0)$, which implies $\mathsf{M} \equiv_\alpha \mathsf{Z}$ for all $\alpha \in \omega_1$ using the rule $(\leq_\alpha)$. Now, consider the set

$$\mathcal{R} = \{ i \mid \mathsf{Z}.R_i \neq \varnothing \} = \{ i \mid \mathsf{N}.R_i \neq \varnothing \}$$

Note that $\mathcal{R} = \{i_1,\dots,i_k\}$ for some $k < \mathsf{Z}.r_0 (= \mathsf{N}.r_0)$. For every $i \in \mathcal{R}$, let $\mathsf{Z}.R_i = a_i$ and $\mathsf{N}.R_i = a'_i$. Also, let $\mathsf{Z}.T = [b_1,\dots,b_m]$ and $\mathsf{N}.T = [b'_1,\dots,b'_m]$. By assumption, $a_i \simeq^{\text{æ}}_{\mathbb{A}} a'_i$ and $b_j \simeq^{\text{æ}}_{\mathbb{A}} b'_j$ for every $i \in \mathcal{R}$, and $j \,(1 \leq j \leq m)$. By induction hypothesis, $\#^{-1}(a_i) \equiv_{\gamma_i} \#^{-1}(a'_i)$ and $\#^{-1}(b_j) \equiv_{\delta_j} \#^{-1}(b'_j)$. Using the rule $(<_\alpha)$, the same holds for $\equiv_\alpha$ setting $\alpha = \sup_{i\in\mathcal{R},1\leq j\leq m}\{\gamma_i,\delta_j\}$. Putting everything together, we obtain:

$$\begin{array}{rcll}
\mathsf{M} & \equiv_\alpha & \mathsf{Z} = \langle \mathsf{Z}.\vec{R}, P, [b_1,\dots,b_m] \rangle & \\
 & \equiv_\alpha & \langle \mathsf{Z}.\vec{R}[R_{i_1} := a'_{i_1}], P, [b_1,\dots,b_m] \rangle, & \text{by } (R_\alpha), \\
 & \equiv_\alpha & \cdots & \vdots \\
 & \equiv_\alpha & \langle \mathsf{Z}.\vec{R}[R_i := a'_i]_{i\in\mathcal{R}}, P, [b_1,\dots,b_m] \rangle, & \text{by } (R_\alpha), \\
 & = & \langle \mathsf{N}.\vec{R}, P, [b_1,\dots,b_m] \rangle, & \text{by definition}, \\
 & \equiv_\alpha & \langle \mathsf{N}.\vec{R}, P, [b'_1,b_2,\dots,b_m] \rangle, & \text{by } (T_\alpha), \\
 & \equiv_\alpha & \cdots & \vdots \\
 & \equiv_\alpha & \langle \mathsf{N}.\vec{R}, P, [b'_1,\dots,b'_m] \rangle, & \text{by } (T_\alpha), \\
 & = & \mathsf{N}, & \text{by definition}.
\end{array}$$

We conclude by applying the transitivity rule (Tr_α) that $\mathsf{M} \equiv_\alpha \mathsf{N}$.

Case (æ). By induction hypothesis, for every $a \in \mathbb{A}$, there exists $\gamma_a \in \omega_1$ such that $\mathsf{M} \mathbin{@} [a] \equiv_{\gamma_a} \mathsf{N} \mathbin{@} [a]$. For $\gamma = \sup_{a\in\mathbb{A}} \gamma_a$, we get $\mathsf{M} \mathbin{@} [a] \equiv_\gamma \mathsf{N} \mathbin{@} [a]$ by $(\leq_\alpha)$. By $(\approx_\alpha)$ we get $\mathsf{M} \approx_\alpha \mathsf{N}$ for $\alpha = \gamma + 1 \in \omega_1$, conclude by $(\subseteq^{\approx}_\alpha)$ and $(\subseteq^{\sim}_\alpha)$.

(Reflexivity), (Symmetry) and (Transitivity) follow from the respective property of $\equiv_\alpha$.

Concerning items (ii)–(v) the implication $(\Leftarrow)$ is trivial. We analyze $(\Rightarrow)$.

(ii) By induction on a derivation of $\mathsf{M} \equiv_0 \mathsf{N}$, using Theorem 3.11.

(iii) By induction on a derivation of $\mathsf{M} \equiv_\alpha \mathsf{N}$.

Case $(\subseteq^{\sim}_\alpha)$. Trivial.

Case (R_α). I.e., $\mathsf{M} = \mathsf{Z}[R_i := a]$, $\mathsf{N} = \mathsf{Z}[R_i := b]$ and $\#^{-1}(a) \equiv_\alpha \#^{-1}(b)$. By induction hypothesis, there exist $c_1, \dots, c_k \in \mathbb{A}$ such that

$$\#^{-1}(a) \sim_\alpha \#^{-1}(c_1) \sim_\alpha \cdots \sim_\alpha \#^{-1}(c_k) = \#^{-1}(b).$$

The case follows by applying the rule $(R^{\sim}_\alpha)$.

Case $(@_\alpha)$. Analogous, by applying $(@^{\sim}_\alpha)$.

Case (T_α). Analogous, by applying $(T^{\sim}_\alpha)$.

Case (Tr_α). Straightforward from the IH.

Case $(\leq_\alpha)$. By IH and $(\leq^{\sim}_\alpha)$.

Cases (Reflexivity), (Symmetry). Straightforward from the IH.

(iv) By induction on a derivation of $\mathsf{M} \sim_\alpha \mathsf{N}$.

Case $(\subseteq^{\approx}_\alpha)$. Take $\mathsf{C} = \xi$.

Case $(R^{\sim}_\alpha)$. I.e., $\mathsf{M} = \mathsf{Z}[R_i := a]$, $\mathsf{N} = \mathsf{Z}[R_i := b]$ and $\#^{-1}(a) \sim_\alpha \#^{-1}(b)$. By induction hypothesis, there exist $\mathsf{C}' \in \mathcal{M}^{\xi}_{\mathbb{X}}$ having address $c = \underline{\#}\mathsf{C}' \in \mathbb{X}$, $\mathsf{M}', \mathsf{N}' \in \mathcal{M}_{\mathbb{A}}$ such that $\mathsf{C}'(\!|\mathsf{M}'|\!) = \#^{-1}(a)$, $\mathsf{C}'(\!|\mathsf{N}'|\!) = \#^{-1}(b)$ and $\mathsf{M}' \approx_\alpha \mathsf{N}'$. We conclude by taking $\mathsf{C} = \mathsf{Z}[R_i := c]$.

Case $(@^{\sim}_\alpha)$. Analogous.

Case $(T^{\sim}_\alpha)$. Take $\mathsf{C} = \mathsf{C}' \,@\, T$, where C' is obtained from the IH.

Case $(\leq^{\sim}_\alpha)$. It follows from the IH, by applying $(\leq^{\sim}_\alpha)$ and $(\leq^{\approx}_\alpha)$.

Cases (Reflexivity), (Symmetry). Straightforward from the IH.

(v) Immediate. □

Consider now a scenario where $\mathsf{C}(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{C}'(\!|\mathsf{M}|\!)$. Assuming $\mathsf{M} \approx_\alpha \mathsf{N}$, one might expect that also $\mathsf{C}(\!|\mathsf{N}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{C}'(\!|\mathsf{N}|\!)$ holds. In general, this is not the case because M and N might reach the head position and get control of the computation. Using the underlined (head-)reduction from Definition 5.4(ii) we can substitute N for M along the reduction (when it comes in head position) and construct a proof of $\mathsf{C}(\!|\mathsf{N}|\!) \equiv_\gamma \mathsf{C}'(\!|\mathsf{N}|\!)$ having a lower ordinal $\gamma < \alpha$.

Lemma 5.9. *Let* $\alpha > 0$, $\mathsf{C} \in \mathcal{M}^{\xi}_{\mathbb{X}}$, $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$ *such that* $\mathsf{M} \approx_\alpha \mathsf{N}$. *If* $\mathsf{C} \to^{\mathsf{M}}_{\mathsf{h}} \mathsf{C}'$ *and* $\mathsf{C}'(\!|\mathsf{M}|\!) \not\twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}()$, *then there exists* $\gamma < \alpha$ *such that* $\mathsf{C}(\!|\mathsf{N}|\!) \equiv_\gamma \mathsf{C}'(\!|\mathsf{N}|\!)$.

Proof. By cases on the shape of C.

Case $\mathsf{C} = \xi \,@\, T$ for some $T \in \mathbb{T}_{\mathbb{X}}$ and $\mathsf{C}' = \mathsf{M} \,@\, T$. From $\mathsf{M} \approx_\alpha \mathsf{N}$ and Lemma 5.8(v), we get that $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}(\mathsf{M}')$ for some $\mathsf{M}' \in \mathcal{M}_{\mathbb{A}}$. Since $\mathsf{C}'(\!|\mathsf{M}|\!) = \mathsf{M} \,@\, (T(\!|\mathsf{M}|\!))$ cannot reduce to a stuck addressing machine, we must have $T(\!|\mathsf{M}|\!) \neq []$. In other words, $T = [a_0, \dots, a_n]$ for some $n \geq 0$. Notice that, for all $a_i \in \mathbb{T}_{\mathbb{X}}$, we have $a_i(\!|\mathsf{N}|\!) \in \mathbb{A}$ (by construction). By Lemma 5.8(v), there exists $\gamma < \alpha$ such that $\mathsf{N} \,@\, [a_0(\!|\mathsf{N}|\!)] \equiv_\gamma \mathsf{M} \,@\, [a_0(\!|\mathsf{N}|\!)]$. By definition:

$$\mathsf{C}(\!|\mathsf{N}|\!) = \mathsf{N} \,@\, T(\!|\mathsf{N}|\!)\,, \text{ and } \mathsf{C}'(\!|\mathsf{N}|\!) = \mathsf{M} \,@\, T(\!|\mathsf{N}|\!)\,.$$

So we construct the proof:

$$\frac{\mathsf{N} \,@\, [a_0(\!|\mathsf{N}|\!)] \equiv_\gamma \mathsf{M} \,@\, [a_0(\!|\mathsf{N}|\!)]}{\mathsf{N} \,@\, [a_0(\!|\mathsf{N}|\!), \dots, a_n(\!|\mathsf{N}|\!)] \equiv_\gamma \mathsf{M} \,@\, [a_0(\!|\mathsf{N}|\!), \dots, a_n(\!|\mathsf{N}|\!)]} \ (T_\gamma)$$

In all the other cases, $\mathsf{C}(\!|\mathsf{N}|\!) \to_h \mathsf{C}'(\!|\mathsf{N}|\!)$, therefore $\mathsf{C}(\!|\mathsf{N}|\!) \equiv_0 \mathsf{C}(\!|\mathsf{N}|\!)$. □

Corollary 5.10. *Let* $n \in \mathbb{N}$, $\alpha > 0$, $\mathsf{C} \in \mathcal{M}^{\xi}_{\mathbb{X}}$, $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$. *If* $\mathsf{C}(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ *and* $\mathsf{M} \approx_\alpha \mathsf{N}$ *then there exists* $\gamma < \alpha$ *such that* $\mathsf{C}(\!|\mathsf{N}|\!) \equiv_\gamma \mathsf{x}_n$.

Proof. Assume $\mathsf{C}(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$. Equivalently, by Lemma 5.6, we have $\mathsf{C} \twoheadrightarrow^{\mathsf{M}}_{\mathsf{h}} \mathsf{x}_n$. By definition, there exists $\mathsf{C}_1, \dots, \mathsf{C}_k \in \mathcal{M}^{\xi}_{\mathbb{X}}$ such that

$$\mathsf{C} = \mathsf{C}_1 \to^{\mathsf{M}}_{h} \cdots \to^{\mathsf{M}}_{h} \mathsf{C}_k = \mathsf{x}_n$$

Notice that $\mathsf{C}_i(\!|\mathsf{M}|\!) \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ and, since $\neg\mathsf{stuck}(\mathsf{x}_n)$, we have $\mathsf{C}_i(\!|\mathsf{N}|\!) \not\twoheadrightarrow_{\mathsf{h}} \mathsf{stuck}()$. By Lemma 5.9, there exists $\gamma_1, \dots, \gamma_k < \alpha$ such that $\mathsf{C}_i(\!|\mathsf{N}|\!) \equiv_{\gamma_i} \mathsf{C}_{i+1}(\!|\mathsf{N}|\!)$. By transitivity ($\mathrm{Tr}_\alpha$) and ($\leq_\alpha$) we obtain $\mathsf{M} \equiv_\alpha \mathsf{x}_n$ for $\alpha = \sup_i \gamma_i$. □

Proposition 5.11. *Let* $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$*,* $\alpha \in \omega_1$ *and* $n \in \mathbb{N}$*. If* $\mathsf{M} \equiv_\alpha \mathsf{N}$ *and* $\mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ *then* $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$*.*

Proof. We proceed by induction on α. Since we perform a double induction, the induction hypothesis with respect to this induction is called the α-IH (α-inductive hypothesis).

Case $\alpha = 0$. By Lemma 5.8(ii), we get $\mathsf{M} \equiv^{\text{æ}}_{\mathbb{A}} \mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$, so we conclude $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ by confluence (Theorem 3.11) and $\to_{\mathsf{i}}$-postponement (Lemma 3.8).

Case $\alpha > 0$. By Lemma 5.8(iii), there exist $\mathsf{Z}_1, \dots, \mathsf{Z}_k$ such that

$$\mathsf{M} \sim_\alpha \mathsf{Z}_1 \sim_\alpha \cdots \sim_\alpha \mathsf{Z}_k = \mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n \tag{5.1}$$

By induction on k, we prove that (5.1) implies $\mathsf{M} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$. We call this k-IH.

Subcase $k = 0$. Then $\mathsf{M} = \mathsf{N} \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$ and we are done.

Subcase $k > 0$. From the k-IH we derive $\mathsf{Z}_1 \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$. From $\mathsf{M} \sim_\alpha \mathsf{Z}_1$ and Lemma 5.8(iv), there is a context-machine C such that $\mathsf{M} = \mathsf{C}[\mathsf{M}']$ and $\mathsf{Z}_1 = \mathsf{C}[\mathsf{N}']$ with $\mathsf{M}' \approx_\alpha \mathsf{N}'$ and $\mathsf{C}[\mathsf{N}'] \twoheadrightarrow_{\mathsf{h}} \mathsf{x}_n$. By applying Lemma 5.9 we obtain $\mathsf{C}[\mathsf{M}'] \equiv_\gamma \mathsf{x}_n$ for some $\gamma < \alpha$. We conclude by applying the α-IH. □

From this proposition, Lemma 4.5(ii) follows by applying Lemma 5.8(i).

6. Conclusions and Further Works

In this paper, we have shown that it is possible to obtain a model of the untyped λ-calculus based on a kind of computational machines that operate exclusively on "addresses", without any reference to some basic data-type. The result only depends on the assumption that every machine has a unique address (and *vice versa* every address identifies a machine) and is completely independent from the specific nature of the addresses themselves.

A natural question that can be raised is whether addressing machines can be seen as a representation of Combinatory Logic's operational semantics in disguise, since their instructions essentially incorporate the contents of the rewriting rules of the basic combinators. To correct this simplistic point of view, observe that the address table map is an arbitrary bijection, whence there are uncountably many possible choices. In particular, address table maps may have arbitrary computational complexity. On the contrary, the operational semantics is constrained to work with the subterms of the current term, i.e. it uses a very "narrow" address table map. We plan to investigate in future works what possibilities arise from the extra degree of freedom given by the arbitrary nature of this map.

We would like to explore whether the theory of the λ-model $\mathcal{S}$ defined in Section 5 depends on the specific nature of the bijection $\#(\cdot) : \mathbb{A} \to \mathcal{M}_{\mathbb{A}}$. As discussed in Remark 2.9, certain ATMs display some peculiarities, since they may create infinite chains of references morally representing infinitary objects. In fact, given an ATM $\#(-)$ and an injection $f : \mathbb{N} \to \mathbb{A}$, a simple application of Hilbert's Hotel allows to define a new ATM $\#'(-)$ where machines $(\mathsf{M}^f_n)_{n\in\mathbb{N}}$ satisfying $\mathsf{M}^f_n = \langle f(n), \varepsilon, [\#'(\mathsf{M}^f_{n+1})]\rangle$ exist. However, these machines are not λ-definable, whence they should simply constitute non-definable "junk" from the model-theoretic perspective. Therefore, we conjecture that $\mathrm{Th}(\mathcal{S})$ is actually independent from the choice of the lookup function $\#(\cdot)$. In case of a positive answer, it would be interesting to provide a complete characterization of the associated λ-theory.

In Section 5 we have shown that $\mathrm{Th}(\mathcal{S})$ is neither extensional nor sensible. This is due to the fact that we kept our construction tight: at each step — from applicative structure, to combinatory algebra, and finally to λ-model — we added the minimal quotient resolving the issue. In order to obtain an extensional model, it would be sufficient to replace the rule (æ) with a form of extensionality non-restricted to machines that become stuck once executed. Similarly, a sensible model can be obtained by collapsing all the addresses of those machines exhibiting a non-terminating behaviour when executed on a number of indeterminates large enough. These quotients are not difficult to define, but the non-trivial problem becomes to prove that the resulting λ-model is non-trivial. This is left for further works.

A different line of research, more in the direction of functional programming, is to expand the computational capabilities of addressing machines by adding simple data-types and the associated basic operations. In fact, although data-types are unnecessary to achieve Turing-completeness, they are desirable to perform arithmetical operations and conditionals. Preliminary investigations [IMM21] show that extending addressing machines with numerals, conditional branching, natural numbers basic arithmetic instructions opens the way for representing Plotkin's `PCF` [Plo77]. These investigations show the precise simulation existing between addressing machine's head reduction and the corresponding evaluation strategy defined on `PCF` extended with explicit substitutions [LM99]. We will check if results of this kind extend to the call-by-value untyped setting. To begin with, we plan to study whether addressing machines can be used to represent the crumbling abstract machines from [ACGC19].

To perform some tests on addressing machines, we have implemented the formalism both in functional and imperative style. Even if the sources remain for internal use only, some technical choices deserve a discussion. Although not explicitly required by the definition, any implementation must rely on a computable association between addressing machines and the corresponding addresses. To implement such a bijection, one could try to use as addresses the actual pointers to the structures representing the machines, but the referenced data might change without affecting the address. A naive solution consists in defining an association list ℓ of type $\mathbb{A} \times \mathcal{M}_{\mathbb{A}}$ and an incremental approach. The list ℓ is initialized as the empty-list. When a new machine M is created, one checks whether M belongs to $\pi_2(\ell)$: in the affirmative case there is nothing to do as the machine is already known; otherwise, a new address a is generated and the pair (a, M) is added to the list ℓ. This guarantees that an address uniquely identifies a machine and that, when an address is used, the corresponding machine has already been introduced. For a more optimized solution one should employ the hash-consing technique, allowing to implement the same concept in a more efficient way.

Acknowledgements. This work is partly supported by ANR Project PPS, ANR-19-CE48-0014. We would like to thank Henk Barendregt for interesting discussions concerning the problem of finding a model of λ-calculus based on recursive functions, as well as the role of the (ω)-rule. We are grateful to the anonymous reviewers for the careful reading and insightful suggestions. We also thank Nicolas Münnik for his comments on the paper.

References

[ACGC19] Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. `doi:10.1145/3354166.3354169`.

[Bar71] Henk Pieter Barendregt. *Some extensional term models for combinatory logics and λ-calculi.* Ph.D. thesis, Utrecht Universiteit, the Netherlands, 1971.

[Bar84] Henk Pieter Barendregt. *The lambda-calculus, its syntax and semantics.* Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, revised edition, 1984.

[BLM05] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005. doi:10.1007/11601548_7.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory logic. Volume I.* Number 1 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 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.

[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.

[HLS72] J. Roger Hindley, Rasmus Lerdorf, and Jonathan P. Seldin. *Introduction to combinatory logic.* Cambridge University Press, 1972.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus.* Cambridge University Press, 1986.

[IMM21] Benedetto Intrigila, Giulio Manzonetto, and Nicolas Münnich. Extended addressing machines for PCF, with explicit substitutions, 2021. Submitted.

[IMP19] Benedetto Intrigila, Giulio Manzonetto, and Andrew Polonsky. Degrees of extensionality in the theory of Böhm trees and Sallé's conjecture. *Log. Methods Comput. Sci.*, 15(1), 2019.

[IS06] Benedetto Intrigila and Richard Statman. Solution of a problem of Barendregt on sensible λ-theories. *Log. Methods Comput. Sci.*, 2(4), 2006.

[IS17] Benedetto Intrigila and Richard Statman. Lambda theories allowing terms with a finite number of fixed points. *Math. Struct. Comput. Sci.*, 27(3):405–427, 2017.

[Kle36] Stephen Cole Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2(2):340 – 353, 1936. doi:10.1215/S0012-7094-36-00227-2.

[Koy82] C.P.J. Karst Koymans. Models of the lambda calculus. *Information and Control*, 52(3):306–332, 1982.

[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.

[LS04] Stefania Lusin and Antonino Salibra. The lattice of lambda theories. *J. Log. Comput.*, 14(3):373–394, 2004.

[Mey82] Albert R. Meyer. What is a model of the lambda calculus? *Information and Control*, 52(1):87–122, 1982.

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

[MPSS19] Giulio Manzonetto, Andrew Polonsky, Alexis Saurin, and Jakob Grue Simonsen. The fixed point property and a technique to harness double fixed point combinators. *J. Log. Comput.*, 29(5):831–880, 2019.

[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.

[Rog87] Hartley Rogers Jr. *Theory of recursive functions and effective computability (Reprint from 1967).* MIT Press, 1987.

[Sel02] Peter Selinger. The lambda calculus is algebraic. *J. Funct. Program.*, 12(6):549–566, 2002.

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

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 41:230–267, 1936.

[Wad76] Christopher P. Wadsworth. The relation between computational and denotational properties for Scott's $\mathcal{D}_\infty$-models of the lambda-calculus. *SIAM J. Comput.*, 5(3):488–521, 1976.