


# Memory trees and Taylor expansion for the $\lambda$ I-calculus

No variables left behind!

Rémy Cerda ✉ 🏠 

Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Giulio Manzonetto ✉ 🏠 

Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Alexis Saurin ✉ 🏠

Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

INRIA  $\pi^3$ , Paris, France

---

## Abstract

Although the  $\lambda$ I-calculus is a natural fragment of the  $\lambda$ -calculus, obtained by forbidding the erasure, its equational theories did not receive much attention. The reason is that all proper denotational models studied in the literature equate all non-normalizable  $\lambda$ I-terms, whence the associated theory is not very informative. The goal of this paper is to introduce a previously unknown theory of the  $\lambda$ I-calculus, induced by a notion of evaluation trees that we call ‘memory trees’. The memory tree of a  $\lambda$ I-term is an annotated version of its Böhm tree, remembering all free variables that are hidden within its meaningless subtrees, or pushed into infinity along its infinite branches.

We develop the associated theories of program approximation: the first approach—more classic—is based on finite trees and continuity, the second adapts Ehrhard and Regnier’s Taylor expansion. We then prove a Commutation Theorem stating that the normal form of the Taylor expansion of a  $\lambda$ I-term coincides with the Taylor expansion of its memory tree. As a corollary, we obtain that the equality induced by memory trees is compatible with abstraction and application. We conclude by discussing the cases of Lévy-Longo and Berarducci trees, and generalizations to the full  $\lambda$ -calculus.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Rewrite systems; Theory of computation  $\rightarrow$  Lambda calculus; Theory of computation  $\rightarrow$  Denotational semantics

**Keywords and phrases**  $\lambda$ -calculus, program approximation, Taylor expansion,  $\lambda$ I-calculus, persistent free variables, Böhm trees, memory trees

## 1 Introduction

In the pioneering article “The calculi of lambda-conversion” [12] Alonzo Church introduced the  $\lambda$ -calculus together with its fragment without weakening, called *the  $\lambda$ I-calculus*, where each abstraction must bind *at least* one occurrence of a variable. Historically, the  $\lambda$ I-calculus has proven to be a useful framework for establishing results such as the finiteness of developments and standardization, which were successfully proven for the full  $\lambda$ -calculus only decades later. Despite that, in the last forty years there are no groundbreaking advances in its study.

The study of models and theories of  $\lambda$ -calculus flourished in the 1970s [4] and remained central in theoretical computer science for decades [5]. Although the equational theories of  $\lambda$ -calculus, called  *$\lambda$ -theories*, form a complete lattice of cardinality  $2^{\aleph_0}$  [31], only a handful are of interest to computer scientists. Most articles focus on  $\lambda$ -theories generated through some notion of ‘evaluation tree’ of a  $\lambda$ -term, which means that two  $\lambda$ -terms are equated in the theory exactly when their evaluation trees coincide. The definitions of evaluation trees present in the literature follow the same pattern: they collect in a possibly infinite tree all stable portions of the output coming out from the computation, and replace the subterms that are considered *meaningless* by a constant  $\perp$ , representing the lack of information. By

modifying the notions of ‘stable’ and ‘meaningless’, one obtains Böhm trees [3], possibly endowed with some form of extensionality [40, 25], Lévy-Longo trees [29, 30] and Berarducci trees [7]. In the modern language of infinitary term rewriting systems, these trees can be seen as the transfinite normal forms of infinitary  $\lambda$ -calculi [21, 13]. Another popular method for defining  $\lambda$ -theories is via observational equivalences [35]: two  $\lambda$ -terms are equivalent if they display the same behaviour whenever plugged in any context. For instance, the theory of Scott’s  $\mathcal{D}_\infty$  model [39] captures the maximal consistent observational equivalence [24, 45].

Since the  $\lambda$ -calculus is a conservative extension of the  $\lambda$ I-calculus, every theory of the former is also a theory of the latter, namely a  $\lambda$ I-theory, while the converse does not hold. A typical example is the  $\lambda$ I-theory  $\mathcal{H}_I$  generated by equating all  $\lambda$ I-terms that are not  $\beta$ -normalizing. The maximal consistent observational equivalence  $\mathcal{H}_I^\eta$  is similar, except for the fact that it is extensional. Note that equating all  $\lambda$ -terms without a  $\beta$ -normal form gives rise to an inconsistent  $\lambda$ -theory, but in the context of  $\lambda$ I-calculus it is consistent and arises naturally: all ‘proper’ denotational models of the  $\lambda$ I-calculus induce either  $\mathcal{H}_I$  or  $\mathcal{H}_I^\eta$  [23]. This is somewhat disappointing, since models and theories are most valuable when they capture non-trivial operational properties of programs, and normalization is relatively elementary in this setting. In this paper we are going to introduce a new (proper)  $\lambda$ I-theory, induced by a notion of evaluation trees inspired by Böhm trees, but explicitly designed for  $\lambda$ I-terms.

**Böhm trees and Taylor expansion for the  $\lambda$ I-calculus.** An important feature of Böhm trees is that, because of their coinductive nature, they are capable of pushing some subterms into infinity. Consider for instance a  $\lambda$ I-term  $M$  satisfying

$$Mx f \rightarrow_\beta f(Mx f) \rightarrow_\beta f(f(Mx f)) \rightarrow_\beta f(f(f(Mx f))) \rightarrow_\beta f^n(Mx f) \rightarrow_\beta \dots$$

then the Böhm tree of  $Mx f$  is the ‘limit’ of this infinite reduction:  $f^\omega = f(f(f(\dots)))$ . Observe that  $x$  is ‘persistent’ in the sense that it is never erased along any finite reduction, but is forgotten at the limit. We say that  $x$  is *pushed into infinity* in the Böhm tree of  $Mx f$ . To some extent, also the variable  $f$  is pushed into infinity, but it occurs infinitely often in  $f^\omega$ . The variable  $x$  can also disappear because it is hidden behind a meaningless term like  $\Omega$ . For instance, the Böhm tree of the  $\lambda$ I-term  $N = \Omega x$  is just  $\perp$  and we say that  $x$  is *left behind*.

As a consequence there are  $\lambda$ I-terms, like  $\lambda x f.M$  and  $\lambda x y.y(\Omega x)$ , whose Böhm tree is not a  $\lambda$ I-tree [4, Def. 10.1.26] since the variable  $x$  is abstracted, but does not appear in the tree. This shows that Böhm trees are not well-suited for the  $\lambda$ I-calculus, and the question of whether other notions of trees can model  $\lambda$ I-terms in a more faithful way naturally arises.

In this paper we introduce a notion of Böhm trees keeping tracks of the variables persistently occurring along each possibly infinite path, or left behind a meaningless subterm. We call them *memory trees* as they remember all the variables present in the terms generating them, even if these variables are never actually used in their evaluations. This is obtained by simply annotating such variables on the branches of the Böhm tree, and on its  $\perp$ -nodes, but bares interesting consequences: since memory trees are invariant under  $\beta$ -conversion, showing that two terms have distinct memory trees becomes a way of separating them modulo  $=_\beta$ .

Once the coinductive definition of memory trees is given (Definition 10), we develop the corresponding theory of program approximation based on finite trees. Perhaps surprisingly, in the finite approximants, the only variable annotations that are actually required are those on the constant  $\perp$ , as all other labels can be reconstructed when taking their supremum. Our main result in this setting is the Approximation Theorem 21 stating that the memory tree of a  $\lambda$ I-term is uniquely determined by the (directed) set of all its finite approximants.

Inspired by quantitative semantics of linear logics, Ehrhard and Regnier proposed a theory of program approximation for regular  $\lambda$ -terms based on Taylor expansion [18, 20]. A  $\lambda$ -term

is approximated by a possibly infinite power-series of programs living in a completely linear *resource calculus*, where no resource can be duplicated or erased along the computation. This theory is linked to Böhm trees by a Commutation Theorem [19] stating that the normal form of the Taylor expansion of a  $\lambda$ -term is equal to the Taylor expansion of its Böhm tree.

The second part of our paper is devoted to explore the question of whether it is possible to define a Taylor expansion for the  $\lambda$ I-calculus capturing the memory tree equality. In Section 4 we design a  $\lambda$ I-resource calculus with memory, mixing linear and non-linear features. Our terms are either applied to non-empty *bags* (finite multiset) of non-duplicable resources, or to an empty bag  $1_X$  annotated with a set  $X$  of variables (the non-linear parts of the terms). The reduction of a resource term  $t$  preserves its free variables  $\text{fv}(t)$ : if the reduction is valid then  $t$  consumes all its linear resources and each variable is recorded in some labels, otherwise an exception is thrown and  $t$  reduces to an empty program  $0_X$ , where  $X = \text{fv}(t)$ .

First we prove that the resource calculus so-obtained is confluent and strongly normalizing, then we use it as the target language of a Taylor expansion with memory, capable of approximating both  $\lambda$ I-terms and memory trees. Our main result is that the Commutation Theorem from [19] extends to this setting: the normal form of the Taylor expansion with memory of a  $\lambda$ I-term always exists and is equal to the Taylor expansion with memory of its memory tree (Theorem 39). As consequences (Corollaries 45 and 46), we obtain that: 1) The equality induced on  $\lambda$ I-terms by memory trees coincides with the equality induced by the normalized Taylor expansion with memory; 2) The equality induced by memory trees is compatible with application and abstraction, in the sense of the  $\lambda$ I-calculus, and it is therefore a  $\lambda$ I-theory. We consider this work a first step in a broader line of research, which we discuss further in the conclusion of the paper. In particular, we aim to extend our formalism to the full  $\lambda$ -calculus and explore its applicability to other kinds of trees.

## Related works

On the syntactic side, the  $\lambda$ I-calculus can be translated into MELL proofnets without weakening studied in [14, 15]. Our memory trees can be presented as labelled Böhm trees, and therefore share similarities with Mellies's (infinitary)  $\lambda$ -terms 'with boundaries', whose branches are annotated by booleans [34]. However, our labels are describing additional points occurring at transfinite positions, a phenomenon reminiscent of the non-monotonic pre-fixed point construction defined in [6]. They can also be seen as an instances of the transfinite terms considered in [28], but simpler, because no reduction is performed at transfinite positions. Thus, memory trees should correspond to normal forms of an infinitary labelled  $\lambda$ I-calculus, just like Böhm trees are the normal forms of the infinitary  $\lambda$ -calculus [27]. We wonder whether a clever translation of the  $\lambda$ I-calculus into the process calculus in [16] would allow us to retrieve the memory trees and the associated Taylor expansion, in which case our Commutation Theorem might become an instance of their general construction.

Concerning denotational semantics, certain *relevant* intersection type systems [42, 22, 1] can be used to represent models of the  $\lambda$ I-calculus, as they describe reflexive objects in the smcc of cpos and strict functions [26]. In [23], the authors show that the model introduced in [17] is fully abstract for the  $\lambda$ I-theory  $\mathcal{H}_1^\eta$ . The results in [15] suggest that the relational semantics, endowed with the comonad of finite *non-empty* multisets, contains models whose theory is either  $\mathcal{H}_1$  or  $\mathcal{H}_1^\eta$ . No filter model or relational graph model can induce the memory tree equality, since they equate all  $\lambda$ -terms having the same Böhm tree [38, 8]. However, the relational model  $\mathcal{E}$  defined in [9] using the comonad of finite multisets with possibly infinite coefficients, gives hope since it separates  $Mxf$  from  $Myf$ , and  $\Omega x$  from  $\Omega y$ , when  $x \neq y$ . This category appears to be the most promising for finding models capturing memory trees.

## 2 Preliminaries

We recall some basic notions and results concerning the  $\lambda$ -calculus, and its fragment without weakening known as the  $\lambda$ I-calculus. For more information, we refer to Barendregt's book [4].

### 2.1 The $\lambda$ -calculus, in a nutshell

Let us fix an infinite set  $\mathcal{V}$  of variables. The set  $\Lambda$  of  $\lambda$ -terms is defined by induction

$$\Lambda \ni M, N ::= x \mid \lambda x.M \mid MN \quad (\text{for } x \in \mathcal{V})$$

We assume that application associates on the left, and has higher precedence than abstraction. E.g.,  $\lambda x.\lambda y.\lambda z.xyz$  stands for  $\lambda x.(\lambda y.(\lambda z.(xy)z))$ . We may write  $\lambda \vec{x}.M$  for  $\lambda x_1 \dots \lambda x_n.M$ , and  $M^n(N)$  for  $M(M(\dots M(N)\dots))$ ,  $n$  times. When  $n = 0$ , we get  $\lambda \vec{x}.M = M^0(N) = M$ .

The set  $\text{fv}(M)$  of *free variables* of  $M$  is defined as usual, and we say that  $M$  is *closed* whenever  $\text{fv}(M) = \emptyset$ . Hereafter,  $\lambda$ -terms are considered modulo  $\alpha$ -conversion (see [4, §2.1]).

► **Example 1.** The  $\lambda$ -terms below are used throughout the paper to construct examples:

$$I := \lambda x.x, \quad K := \lambda xy.x, \quad F := \lambda xy.y, \quad B := \lambda fgx.f(g(x)), \quad D := \lambda x.xx, \quad \Omega := DD.$$

The  $\lambda$ -term  $I$  is the identity,  $K, F$  are the projections,  $B$  is the composition,  $D$  is the diagonal and  $\Omega$  a looping term. The *pairing* of  $M, N$  is encoded as  $[M, N] := \lambda x.xMN$ , for  $x$  fresh.

The  $\beta$ -reduction  $\rightarrow_\beta$  is generated by the rule  $(\lambda x.M)N \rightarrow M[N/x]$ , where  $(\lambda x.M)N$  is called a  $\beta$ -redex and  $M[N/x]$  stands for the *capture-free substitution* of  $N$  for all free occurrences of  $x$  in  $M$ . We let  $\rightarrow_\beta$  (resp.  $=_\beta$ ) be the reflexive-transitive (and symmetric) closure of  $\rightarrow_\beta$ . We say that  $M$  is in  $\beta$ -normal form ( $\beta$ -nf) if  $M$  does not contain any  $\beta$ -redex, and that  $M$  has a  $\beta$ -nf if  $M \rightarrow_\beta N$ , for some  $N$  in  $\beta$ -nf. Since  $\rightarrow_\beta$  is confluent, such  $N$  must be unique.

► **Remark 2.** Any  $\lambda$ -term  $M$  can be written in one of the following forms: either  $M = \lambda \vec{x}.yM_1 \dots M_k$ , in which case  $M$  is called a *head normal form* (hnf) and  $y$  its *head variable*; or  $M = \lambda \vec{x}.(\lambda y.P)QM_1 \dots M_k$ , where  $(\lambda y.P)Q$  is referred to as its *head redex*.

The *head reduction*  $\rightarrow_h$  is the restriction of  $\rightarrow_\beta$  obtained by contracting the head redex.

► **Definition 3.** A  $\lambda$ -term  $Y$  is a *fixed-point combinator* (fpc) if it satisfies  $Yx =_\beta x(Yx)$ .

We recall Curry's combinator  $Y$  and define, for all  $l \in \mathcal{V}$ , Polonsky's  $\Theta_l$  and Klop's  $\mathbf{\Theta}_l$  [33].

$$D_f := \lambda x.f(xx), \quad Y := \lambda f.D_f D_f, \quad V := \lambda vlf.f(vvl f), \quad \Theta_l := VVl, \quad \mathbf{\Theta}_l := \lambda e.BYB e l. \quad (1)$$

It is easy to check that  $Y$  and  $\Theta_l$  are fpcs. Regarding the *Bible*  $\mathbf{\Theta}_l$ , which owes its name to its distinctive spelling, we get  $\mathbf{\Theta}_l x =_\beta BYB x l =_\beta Y(Bx)l =_\beta Bx(Y(Bx))l =_\beta x(Y(Bx)l) =_\beta x(\mathbf{\Theta}_l x)$ . Note that the variable  $l$  remains in a passive position along the reduction of  $\mathbf{\Theta}_l$  (and of  $\Theta_l$ ), therefore  $\mathbf{\Theta}_N := \mathbf{\Theta}_l[N/l]$  (resp.  $\Theta_N := \Theta_l[N/l]$ ) remains an fpc, for all  $N \in \Lambda$ .

Although fpcs  $Y$  are not  $\beta$ -normalizable, they differ from  $\Omega$  since they produce increasing stable amounts of information along reduction:  $Y \rightarrow_\beta \lambda f.fY_1 \rightarrow_\beta \lambda f.f^n(Y_n) \rightarrow_\beta \dots$ . Barendregt introduced a notion of 'evaluation tree' to capture this infinitary behaviour [3].

► **Definition 4.** The Böhm tree  $\text{BT}(M)$  of a  $\lambda$ -term  $M$  is *coinductively* defined as follows.

1. If  $M$  has a hnf, that is  $M \rightarrow_h \lambda x_1 \dots \lambda x_n.yM_1 \dots M_k$ , then

$$\text{BT}(M) := \begin{array}{c} \lambda x_1 \dots \lambda x_n.y \\ \swarrow \quad \searrow \\ \text{BT}(M_1) \quad \dots \quad \text{BT}(M_k) \end{array}$$

2.  $\text{BT}(M) := \perp$ , otherwise. The constant  $\perp$  represents the absolute lack of information.

► **Example 5.** Using the terms from Example 1 and (1), let us construct some Böhm trees.

- (i) If  $M$  has a  $\beta$ -normal form  $N$  then, up to isomorphism,  $\text{BT}(M) = N$ . E.g.,  $\text{BT}(\text{DI}) = \text{I}$ .
- (ii) Since  $\Omega$  and  $\text{YK}$  have no hnf, we get  $\text{BT}(\Omega) = \text{BT}(\text{YK}) = \perp$ .
- (iii) If  $Y$  is an fpc, then  $\text{BT}(Y) = \lambda f.f(f(f(\dots)))$ . Thus,  $\text{BT}(\text{Y}) = \text{BT}(\mathbf{\Xi}_l) = \text{BT}(\Theta_l)$ .
- (iv) Using the fpc  $\text{Y}$ , one can define  $\lambda$ -terms  $\text{E}_x, \text{O}_x$  satisfying  $\text{E}_x =_\beta [\Omega, [\Omega x, \text{E}_x]]$  and  $\text{O}_x =_\beta [\Omega x, [\Omega, \text{O}_x]]$ . These two  $\lambda$ -terms both construct a ‘stream’  $[M_1, [M_2, [M_3, [\dots]]]]$ , but the former places  $\Omega x$  at even positions and  $\Omega$  at odd ones, while the latter does the converse. This difference is however forgotten in their Böhm trees:

$$\text{BT}(\text{E}_x) = \text{BT}(\text{O}_x) = [\perp, [\perp, [\perp, [\perp, [\dots]]]] \quad (\text{inline depiction of a tree})$$

- (v) Check that, given  $\text{R} = \lambda y l x.x(y(xl))$ , we have  $\text{BT}(\text{YRl}) = \lambda x_0.x_0(\lambda x_1.x_1(\lambda x_2.x_2(\dots)))$ .

The Böhm tree equality induces an equivalence on  $\lambda$ -terms:  $M =_{\mathcal{B}} N$  iff  $\text{BT}(M) = \text{BT}(N)$ . As shown in [29, 46],  $\mathcal{B}$  is a  $\lambda$ -theory because it contains  $=_\beta$  and is *compatible* with abstraction and application:  $M =_{\mathcal{B}} M'$  entails  $\lambda x.M =_{\mathcal{B}} \lambda x.M'$ ,  $MN =_{\mathcal{B}} M'N$  and  $NM =_{\mathcal{B}} NM'$ .

## 2.2 The $\lambda\text{I}$ -calculus

The set  $\Lambda_1$  of  $\lambda\text{I}$ -terms is the subset of  $\Lambda$  consisting of those  $\lambda$ -terms in which every abstraction binds *at least* one occurrence of the abstracted variable. Note that  $\rightarrow_\beta$  induces a reduction on  $\Lambda_1$  since  $M \in \Lambda_1$  and  $M \rightarrow_\beta N$  entail  $N \in \Lambda_1$ . In other words, the property of being a  $\lambda\text{I}$ -term is preserved by  $\rightarrow_\beta$ . Similarly, the  $\lambda\text{I}$ -calculus inherits from  $\lambda$ -calculus all the notions previously defined like head reduction, Böhm trees, etc. For convenience, we consider an alternative definition of  $\Lambda_1$  simultaneously defining  $\lambda\text{I}$ -terms and their sets of free variables.

► **Definition 6.** (i) For all  $X \subseteq \mathcal{V}$ ,  $\Lambda_1(X)$  is defined as the smallest subset of  $\Lambda$  such that:

$$\frac{}{x \in \Lambda_1(\{x\})} \quad \frac{M \in \Lambda_1(X) \quad x \in X}{\lambda x.M \in \Lambda_1(X - \{x\})} \quad \frac{M \in \Lambda_1(X) \quad N \in \Lambda_1(Y)}{MN \in \Lambda_1(X \cup Y)}$$

- (ii) Finally, the set of all  $\lambda\text{I}$ -terms is given by the disjoint union  $\Lambda_1 = \coprod_{X \subseteq \mathcal{V}} \Lambda_1(X)$ .

► **Remark 7.** (i) Note that  $M \in \Lambda_1(X)$  means that  $M$  is a  $\lambda\text{I}$ -term such that  $\text{fv}(M) = X$ .

As a consequence, if  $M \in \Lambda_1(X)$  then the set  $X$  must be finite.

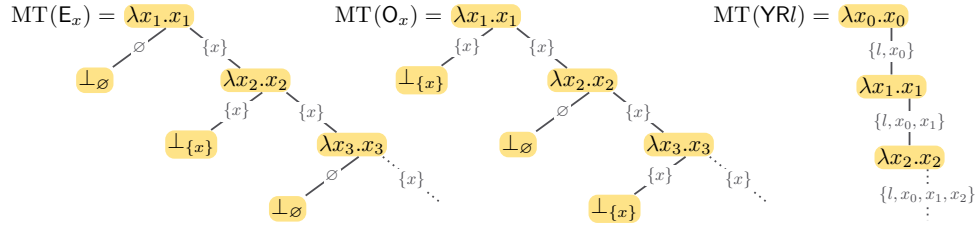
- (ii) Each set  $\Lambda_1(X)$  is closed under  $\beta$ -conversion.
- (iii) Our examples  $\text{I}, \text{B}, \text{D}, \Omega, \text{Y}, \mathbf{\Xi}_l, \Theta_l, \text{E}_x, \text{O}_x, \text{YRl}$  are  $\lambda\text{I}$ -terms, while  $\text{K}, \text{F}, \text{YK} \notin \Lambda_1$ .

► **Definition 8.** A  $\lambda\text{I}$ -theory  $\mathcal{T}$  is given by an equivalence  $=_{\mathcal{T}} \subseteq \Lambda_1^2$  containing  $\beta$ -conversion and compatible with application and abstraction. In the context of  $\lambda\text{I}$ -calculus, the compatibility with abstraction becomes:  $M =_{\mathcal{T}} M'$  and  $x \in \text{fv}(M) \cap \text{fv}(M')$  imply  $\lambda x.M =_{\mathcal{T}} \lambda x.M'$ .

The only subtlety in the definition above stands in the side condition on the compatibility with abstraction. However, this small difference has significant consequences: e.g., the theory axiomatized by equating all terms without a  $\beta$ -nf is consistent as a  $\lambda\text{I}$ -theory, but inconsistent as a  $\lambda$ -theory. The following theorem collects the main properties of the  $\lambda\text{I}$ -calculus.

- **Theorem 9.** (i) Every computable numerical function is  $\lambda\text{I}$ -definable [4, Thm. 9.2.16].
- (ii) Strong and weak  $\beta$ -normalization coincide for the  $\lambda\text{I}$ -calculus [4, Thm. 11.3.4].
- (iii) Every  $\lambda$ -theory is also a  $\lambda\text{I}$ -theory, while the converse does not hold.

By (iii), Böhm trees induce a  $\lambda\text{I}$ -theory, but we argue that this theory does not respect the spirit of  $\Lambda_1$ . E.g., the variable  $l$  is never erased along the reduction of, say,  $\mathbf{\Xi}_l$ , but it disappears in its Böhm tree. Using the terminology of [4], one says that  $l$  is *pushed to infinity* in  $\text{BT}(\mathbf{\Xi}_l)$ . This shows that the Böhm tree of the  $\lambda\text{I}$ -term  $\lambda l.\mathbf{\Xi}_l \in \Lambda_1$  is not a  $\lambda\text{I}$ -tree: the outer  $\lambda$ -abstraction ‘ $\lambda l$ ’ does not bind any free occurrence of  $l$  in  $\text{BT}(\mathbf{\Xi}_l) = \lambda f.f(f(f(\dots)))$ .



**Figure 1** Examples of memory trees of  $\lambda$ I-terms.

### 3 Memory trees

We have seen that Böhm trees are not well-suited for representing  $\lambda$ I-terms faithfully, because variables that are never erased along the reduction are forgotten (i.e., pushed into infinity). We now introduce a notion of evaluation tree that records all variables that are pushed along each path, ensuring that none are forgotten, whether they remain in passive position or not.

### 3.1 The coinductive definition

For every finite set  $X \subseteq \mathcal{V}$ , written  $X \subseteq_f \mathcal{V}$ , we introduce a constant  $\perp_X$  representing any looping  $M \in \Lambda_I(X)$ . Intuitively, the *memory tree* of a  $\lambda$ -term  $M$  is obtained from  $\text{BT}(M)$  by annotating each subtree (also  $\perp$ ) with the free variables of the  $\lambda$ -term that generated it.

► **Definition 10.** *The memory tree of a term  $M \in \Lambda_1(X)$  is coinductively defined as follows:*

- (i) If  $M \twoheadrightarrow_h \lambda x_1 \dots x_n. y M_1 \dots M_k$  with  $y \in X \cup \{\vec{x}\}$ ,  $M_i \in \Lambda_1(X_i)$  (for  $1 \leq i \leq k$ ), then

$$\text{MT}(M) := \begin{array}{c} \lambda x_1 \dots x_n. y \\ \swarrow \quad \searrow \\ \text{MT}(M_1) \quad \dots \quad \text{MT}(M_k) \end{array}$$

- (ii)  $\text{MT}(M) := \perp_X$ , otherwise. Recall that  $X = \text{fv}(M)$ , since  $M \in \Lambda_1(X)$ .

We sometimes use an inline presentation of memory trees, by introducing application symbols annotated with the sets  $X_i$ , as in  $\lambda x_1 \dots x_n. y \cdot^{X_1} T_1 \cdot \dots \cdot^{X_k} T_k$ .

► **Example 11.** (i)  $\text{MT}(\text{I}) = \lambda x.x$ ,  $\text{MT}(\text{B}) = \lambda x.x \cdot^{\{x\}} x$ ,  $\text{MT}(\text{B}) = \lambda f g x.f \cdot^{\{g,x\}} (g \cdot^{\{x\}} x)$ .

(ii)  $\text{MT}(\Omega) = \text{MT}(\lambda y.\Omega y) = \perp_{\emptyset}$ ,  $\text{MT}(\Omega x) = \perp_{\{x\}}$ . More generally:  $\text{MT}(\Omega \vec{x}) = \perp_{\{x_1, \dots, x_n\}}$ .

(iii)  $\text{MT}(\mathbf{Y}) = \lambda f.f.\{f\} (f.\{f\} (f.\{f\} (\dots)))$ . In  $\text{MT}(\underline{\mathbf{I}}_l)$  the variable  $l$  becomes visible:

(iv)  $\text{MT}(\boxplus_l) = \text{MT}(\Theta_l) = \lambda f.f.\{f,l\} (f.\{f,l\} (f.\{f,l\} (\dots)))$ .

(v) Other examples of memory trees are depicted in Figure 1. The justification behind  $\text{MT}(\text{YRl})$  is that  $\text{YRl} =_{\beta} \lambda x_0.x_0((\text{YR})(x_0l)) =_{\beta} \lambda x_0.x_0(\lambda x_1.x_1((\text{YR})(x_1(x_0l)))) =_{\beta} \dots$ .

► **Lemma 12.** *If  $M \in \Lambda_l$  has a  $\beta$ -nf  $N$ , then there is an isomorphism  $\text{MT}(M) \cong N$ .*

**Proof.** Intuitively, since  $N$  is finite, it is possible to reconstruct all the labels in  $\text{MT}(M)$ . Formally, proceed by induction on the structure of  $\text{MT}(M)$  for  $(\Rightarrow)$ , and of  $N$  for  $(\Leftarrow)$ .  $\blacktriangleleft$

► Remark 13. (i) For all *closed* fpcs  $Y \in \Lambda_I$ , we have  $\text{MT}(Y) = \text{MT}(\mathbf{Y}) \cong \text{BT}(\mathbf{Y})$ .

(ii) We have seen  $\text{MT}(\boxplus_l) = \text{MT}(\Theta_l)$ , while  $\text{MT}(\Upsilon) \neq \text{MT}(\boxplus_l) \neq \text{MT}(\Theta_x)$  when  $x \neq l$ .

(iii) The memory trees of  $E_x$  and  $O_x$  are now distinguished:  $MT(E_x) \neq MT(O_x)$ .

**(iv)** The interest of MT(YR) is that it pushes into infinity an increasing amount of variables.

So—at the limit—infinitely many variables are pushed into infinity (but only  $l$  is free).

We are going to show that memory trees are invariant under  $\beta$ -reduction (Prop. 20 plus Thm. 21), and that the equality induced on  $\lambda$ I-terms is a  $\lambda$ I-theory (Corollary 46).

### 3.2 The associated approximation theory

We introduce a theory of program approximation that captures memory trees, by adapting the well-established approach originally developed for Böhm trees [29, 46] (see also [4, Ch. 14]). We start by defining the approximants with memory, corresponding to finite memory trees.

► **Definition 14.** (i) *The set of approximants with memory is  $\mathcal{A}_m := \coprod_{X \subseteq_f \mathcal{V}} \mathcal{A}_m(X)$ , where  $\mathcal{A}_m(X)$  is defined by (for  $A_i \in \mathcal{A}_m(X_i)$ ,  $X_i \subseteq X$ ,  $y \in X \cup \{\vec{x}\}$  and  $\vec{x} \in \bigcup_{i=1}^k X_i$ ):*

$$\mathcal{A}_m(X) \ni A ::= \perp_X \mid \lambda x_1 \dots x_n. y A_1 \dots A_k$$

(ii) *We define  $\sqsubseteq \subseteq \mathcal{A}_m^2$  as the least partial order closed under the following rules:*

$$\frac{A \in \mathcal{A}_m(X) \quad \perp_X \sqsubseteq A}{\perp_X \sqsubseteq A} \quad \frac{A_i \sqsubseteq A'_i \quad A_i, A'_i \in \mathcal{A}_m(X_i) \quad \text{for all } 1 \leq i \leq k \quad \vec{x} \in \{y\} \cup \left(\bigcup_{i=1}^k X_i\right)}{\lambda x_1 \dots x_n. y A_1 \dots A_k \sqsubseteq \lambda x_1 \dots x_n. y A'_1 \dots A'_k}$$

Each  $(\mathcal{A}_m(X), \sqsubseteq)$  is a pointed poset (partially ordered set) with bottom element  $\perp_X$ , while the poset  $(\mathcal{A}_m, \sqsubseteq)$  is not pointed because it has countably many minimal elements. Note that we do not annotate the application symbol in  $A \in \mathcal{A}_m$ , since the labels in its  $\perp$ -nodes carry enough information to reconstruct the finite memory tree associated with  $A$ .

► **Lemma 15.** *There is a bijection between  $\mathcal{A}_m$  and the set of finite memory trees of  $\lambda$ I-terms.*

We now give the recipe to compute the set of approximants with memory of a  $\lambda$ I-term.

► **Definition 16.** (i) *The direct approximant of  $M \in \Lambda_1(X)$  is given by:*

$$\omega_m(M) := \begin{cases} \lambda x_1 \dots x_n. y \omega_m(M_1) \dots \omega_m(M_k), & \text{if } M = \lambda x_1 \dots x_n. y M_1 \dots M_k, \\ \perp_X, & \text{otherwise.} \end{cases}$$

(ii) *The set  $\text{App}_m(M)$  of approximants with memory of  $M \in \Lambda_1$  is defined by:*

$$\text{App}_m(M) := \{A \in \mathcal{A}_m \mid \exists N \in \Lambda_1. M \twoheadrightarrow_\beta N \text{ and } A \sqsubseteq \omega_m(N)\}$$

► **Remark 17.** For all  $M \in \Lambda_1(X)$ , we have  $\omega_m(M) \in \mathcal{A}_m(X)$  and  $\text{App}_m(M) \subseteq \mathcal{A}_m(X)$ .

► **Example 18.** (i)  $\text{App}_m(D) = \{\perp_\emptyset, \lambda x. x \perp_\emptyset, D\}$ ,  $\text{App}_m(\Omega) = \{\perp_\emptyset\}$ ,  $\text{App}_m(\Omega x) = \{\perp_{\{x\}}\}$ .  
(ii)  $\text{App}_m(Y) = \{\perp_\emptyset\} \cup \{\lambda f. f^n(\perp_\emptyset) \mid n \in \mathbb{N}\}$ ,  $\text{App}_m(\mathbf{I}_l) = \{\perp_{\{l\}}\} \cup \{\lambda f. f^n(\perp_{\{l\}}) \mid n \in \mathbb{N}\}$ .  
(iii)  $\text{App}_m(YRl) = \{\perp_{\{l\}}, \lambda x_0. x_0(\perp_{\{x_0, l\}}), \lambda x_0. x_0(\lambda x_1. x_1(\perp_{\{x_0, x_1, l\}})), \dots\}$ .

► **Lemma 19.** (i) *For every  $M, N \in \Lambda_1$ ,  $M \twoheadrightarrow_\beta N$  entails  $\omega_m(M) \sqsubseteq \omega_m(N)$ .*

(ii) *For all  $M \in \Lambda_1(X)$ ,  $\text{App}_m(M)$  is an ideal of the poset  $(\mathcal{A}_m(X), \sqsubseteq)$ . More precisely:*

- a. *non-emptiness:*  $\perp_X \in \text{App}_m(M)$ ;
- b. *downward closure:*  $A \in \text{App}_m(M)$  and  $A' \sqsubseteq A$  imply  $A' \in \text{App}_m(M)$ ;
- c. *directedness:*  $\forall A_1, A_2 \in \text{App}_m(M), \exists A_3 \in \text{App}_m(M)$  such that  $A_1 \sqsubseteq A_3 \sqsupseteq A_2$ .

**Proof.** (i) By an easy induction on the structure of  $M \in \Lambda_1(X)$ , using Remarks 2 and 7.

(ii) (a) and (b) are trivial. For (c), define  $\sqcup$  by setting:  $\perp_X \sqcup A = A \sqcup \perp_X = A$ ,  $x \sqcup x = x$ , if  $A_1, A'_1$  and  $A_2, A'_2$  are compatible, then  $(\lambda x. A_1) \sqcup (\lambda x. A'_1) = \lambda x. (A_1 \sqcup A'_1)$  and  $A_1 A_2 \sqcup A'_1 A'_2 = (A_1 \sqcup A'_1)(A_2 \sqcup A'_2)$ . Check that  $A_3 = A_1 \sqcup A_2$  is their supremum. ◀

The next result shows that  $\beta$ -convertible terms share the same the set of approximants.

► **Proposition 20.** *Let  $M, N \in \Lambda_1(X)$ . If  $M \twoheadrightarrow_\beta N$ , then  $\text{App}_m(M) = \text{App}_m(N)$ .*

**Proof.** By confluence of  $\twoheadrightarrow_\beta$ , using Lemma 19(i). ◀

► **Theorem 21.** *There is a bijection  $\mathcal{A}(\cdot)$  between the set of memory trees and the set of approximants of  $\lambda$ I-terms satisfying  $\mathcal{A}(\text{MT}(M)) = \text{App}_m(M)$ , for all  $M \in \Lambda_1$ .*

**Proof.** From  $\text{MT}(M)$  to  $\text{App}_m(M)$ . Let  $\text{MT}(M)^*$  be the set of all finite subtrees of  $\text{MT}(M)$ , where each truncated subtree  $T$  is replaced by  $\perp_{\text{fv}(T)}$ . By Lemma 15,  $\text{MT}(M)^* \cong \text{App}_m(M)$ .

From  $\text{App}_m(M)$  to  $\text{MT}(M)$ . For every finite path  $\sigma$  in  $\text{MT}(M)$ , there is an  $A \in \text{App}_m(M)$  representing  $\text{MT}(M)$  along  $\sigma$ . Conclude since  $\text{MT}(M)$  is the supremum of all such  $\sigma$ . ◀

## 4 A resource calculus with memory

In this section we introduce the  $\lambda$ I-resource calculus refining the (finite) resource calculus [18], best known as the target language of Ehrhard and Regnier's Taylor expansion [20]. So, the resource calculus is not meant to be a stand-alone language, but rather another theory of approximations for the  $\lambda$ -calculus. Before going further, we recall its main properties.

We consider here the promotion-free fragment of the resource calculus introduced in [37]. Its syntax is similar to the  $\lambda$ -calculus, except for the applications that are of shape  $s\bar{t}$ , where  $\bar{t} = [t_1, \dots, t_n]$  is a multiset of resources called *bag*. The resources populating  $\bar{t}$  are linear as they cannot be erased or copied by  $s$ , they must be used *exactly once* along the reduction. When contracting a term of the form  $s = (\lambda x.s')[t_1, \dots, t_n]$  there are two possibilities.

1. If the number of occurrences of  $x$  in  $s'$  is exactly  $n$ , then each occurrence is substituted by a different  $t_i$ . Since the elements in the bag are unordered, there is no canonical bijection between the resources and the occurrences of  $x$ . The solution consists in collecting all possibilities in a formal sum of terms, the sum representing an inner-choice operator.
2. If there is a mismatch between the number of occurrences and the amount of resources, then  $s$  reduces to the empty-sum,  $\mathbf{0}$ . From a programming-language perspective, this can be thought of as a program terminating abruptly after throwing an uncaught exception. The first-class citizens of the resource calculus are therefore *finite sums* of resource terms, that are needed to ensure the (strong) confluence of reductions. Another important property is strong normalization, that follows from the fact that no resource can be duplicated.

### 4.1 $\lambda$ I-resource expressions and $\lambda$ I-resource sums

Our version of the resource calculus is extended with labels representing the memory of free variables that were present in the  $\lambda$ I-term they approximate. Just like in Definition 14(i) we endowed the constant  $\perp$  with a finite set  $X$  of variables, here we annotate:

- the empty bag  $\mathbf{1}$  of resource terms, since an empty bag of approximants of  $M$  should remember the free variables of  $M$ ;
- the empty sum  $\mathbf{0}$  of resource terms. Indeed, if a resource term vanishes during reduction because of the mismatch described above, its free variables should be remembered.

► **Definition 22.** (i) *For all  $X \subseteq_f \mathcal{V}$ , the sets  $\Delta_1(X)$  and  $!\Delta_1(X)$  are defined by induction:*

$$\begin{array}{c} \frac{}{x \in \Delta_1(\{x\})} \quad \frac{s \in \Delta_1(X) \quad x \in X}{\lambda x.s \in \Delta_1(X - \{x\})} \quad \frac{s \in \Delta_1(X) \quad \bar{t} \in !\Delta_1(Y)}{s\bar{t} \in \Delta_1(X \cup Y)} \\[10pt] \frac{}{1_X \in !\Delta_1(X)} \quad \frac{t_0 \in \Delta_1(X) \quad \dots \quad t_n \in \Delta_1(X)}{[t_0, \dots, t_n] \in !\Delta_1(X)} \end{array}$$

(ii) *The set  $\Delta_1$  of  $\lambda$ I-resource terms and the set  $!\Delta_1$  of bags are given by:*

$$\Delta_1 := \coprod_{X \subseteq_f \mathcal{V}} \Delta_1(X) \quad \text{and} \quad !\Delta_1 := \coprod_{X \subseteq_f \mathcal{V}} !\Delta_1(X).$$

As a matter of notation, we let  $(!)\Delta_I$  denote either  $\Delta_I$  or  $!\Delta_I$ , indistinctly but coherently. We call *resource expressions* generic elements  $s, t \in (!)\Delta_I$ . We denote the union of two bags  $\bar{t}, \bar{u} \in !\Delta_I(X)$  multiplicatively by  $\bar{t} \cdot \bar{u}$ , whose neutral element is the empty bag,  $1_X$ .

- **Remark 23.** (i) In every  $\lambda I$ -resource term of the form  $\lambda x.s$ , the variable  $x$  must occur freely in  $s$ : it may appear in the undecorated underlying term, or in the ‘memory’  $X$  decorating  $1_X$ . Therefore, it makes sense to define  $\text{fv}(s) := X$  whenever  $s \in (!)\Delta_I(X)$ .
- (ii) Each set  $!\Delta_I(X)$  is isomorphic to the monoid of multisets of elements of  $\Delta_I(X)$ . Notice that  $!\Delta_I$  is *not* the set of all bags of elements of  $\Delta_I$ , just its subset of bags whose elements *have the same free variables* (and so inductively in the subterms).
- **Example 24.** (i) The identity  $I$  belongs to  $\Delta_I$ , whereas the projections do not:  $K, F \notin \Delta_I$ .
- (ii) Note that  $\lambda xy.x1_\emptyset \notin \Delta_I$  since  $y \notin \text{fv}(x1_\emptyset)$ , but  $\lambda xy.x1_{\{y\}} \in \Delta_I$  because  $y \in \{y\}$ .
- (iii) The terms  $D_0 = \lambda x.x1_{\{x\}}$  and  $D_{n+1} = \lambda x.x[x, \dots, x]$ , where the bag contains  $n+1$  occurrences of  $x$ , are  $\lambda I$ -resource terms. By Remark 23(ii), we obtain:
- (iv)  $[x, y] \notin !\Delta_I$ , while  $[x, x[x], x[\lambda y.y, \lambda y.y[y]]], (\lambda y.y)[x], \lambda y.y1_{\{x\}} \in !\Delta_I(\{x\}) \subseteq !\Delta_I$ .

We consider resource expressions up to  $\alpha$ -equivalence, under the proviso that abstractions bind linear occurrences of variables as well as occurrences in the memory of empty bags. For instance,  $\lambda xyz.x1_{\{x\}}1_{\{x,y,z\}}$  and  $\lambda x'y'z.x'1_{\{x'\}}1_{\{x',y',z\}}$  are considered  $\alpha$ -equivalent.

► **Definition 25.** For all  $X \subseteq_f \mathcal{V}$ , the set  $\mathbb{N}[(!)\Delta_I(X)]$  of sums of  $\lambda I$ -resource terms (‘resource sums’, for short) is defined as the  $\mathbb{N}$ -semimodule of finitely supported formal sums of expressions in  $(!)\Delta_I(X)$ , with coefficients in  $\mathbb{N}$ . Explicitly, it can be presented as:

$$\mathbb{N}[(!)\Delta_I(X)] \ni \mathbf{s}, \mathbf{t} ::= \mathbf{0}_X \mid r \mid r + \mathbf{s} \quad (\text{for } r \in (!)\Delta_I(X))$$

quotiented by associativity and commutativity of  $+$ , as well as neutrality of  $\mathbf{0}_X$ .

Note that resource expressions are assimilated to the corresponding one-element sum. The constructors of the calculus are extended to resource sums by (bi)linearity, *i.e.* for  $s \in \Delta_I(X)$ ,  $\mathbf{s} \in \mathbb{N}[\Delta_I(X)]$ ,  $\bar{t} \in !\Delta_I(Y)$ ,  $\bar{\mathbf{t}} \in \mathbb{N}[!\Delta_I(Y)]$ ,  $u \in \Delta_I(Y)$  and  $\mathbf{u} \in \mathbb{N}[\Delta_I(Y)]$ , we have:

$$\begin{aligned} (s + \mathbf{s})\bar{t} &:= s\bar{t} + \mathbf{s}\bar{t}, & \mathbf{0}_X\bar{t} &:= \mathbf{0}_{X \cup Y}, & \lambda x.(s + \mathbf{s}) &:= \lambda x.s + \lambda x.\mathbf{s}, & \lambda x.\mathbf{0}_X &:= \mathbf{0}_{X - \{x\}}, \\ s(\bar{t} + \bar{\mathbf{t}}) &:= s\bar{t} + s\bar{\mathbf{t}}, & \mathbf{s}\mathbf{0}_Y &:= \mathbf{0}_{X \cup Y}, & [u + \mathbf{u}] \cdot \bar{t} &:= [u] \cdot \bar{t} + [\mathbf{u}] \cdot \bar{t}, & [\mathbf{0}_Y] \cdot \bar{t} &:= \mathbf{0}_Y. \end{aligned}$$

Therefore, if  $\mathbf{0}_X$  occurs in  $s$  not as a summand but as a proper subterm, then  $s = \mathbf{0}_{\text{fv}(s)}$ .

## 4.2 Memory substitution and resource substitution

While the usual finite resource calculus is completely linear, the variables we store in the memory of  $1_X$  and  $\mathbf{0}_X$  are not. The memory  $X$  remembers the variables present in  $X$  not their amounts, this is the reason why it is modelled as a set, not as a multiset. This consideration leads us to define two kinds of substitutions.

The (non-linear) *memory substitution* of a set  $Y \subseteq_f \mathcal{V}$  for a variable  $x$  in a  $\lambda I$ -resource term  $s$  does not interact with the linear occurrences of  $x$  in  $s$  (*i.e.*, they remain unchanged), it just replaces the ‘memory’ of  $x$  in the empty bags with the content of  $Y$ .

- **Definition 26.** (i) For all  $X, Y \subseteq_f \mathcal{V}$  and  $x \in \mathcal{V}$ , define

$$X \{Y/x\} := \begin{cases} X - \{x\} \cup Y, & \text{if } x \in X, \\ X, & \text{otherwise.} \end{cases}$$

(ii) Given  $s \in (!)\Delta_I(X)$ , the memory substitution of  $x$  by  $Y \subseteq_f \mathcal{V}$  in  $s$  is the resource term  $s\{Y/x\}$  defined as follows (for  $x \neq y$  and, in the abstraction case,  $y \notin Y$ ):

$$\begin{aligned} x\{Y/x\} &:= x, & 1_X\{Y/x\} &:= 1_{X\{Y/x\}}, & (st)\{Y/x\} &:= (s\{Y/x\})(\bar{t}\{Y/x\}), \\ y\{Y/x\} &:= y, & (\lambda y.s)\{Y/x\} &:= \lambda y.s\{Y/x\}, & ([s] \cdot \bar{t})\{Y/x\} &:= [s\{Y/x\}] \cdot (\bar{t}\{Y/x\}). \end{aligned}$$

We now define the *resource substitution* of a bag  $\bar{u}$  for  $x$  in  $s$ , whose effect is twofold. 1) It non-deterministically replaces each linear occurrence of  $x$  with a resource from the bag (as usual). 2) It applies the memory substitution of  $x$  by  $\text{fv}(\bar{u})$  to the resulting sum of terms.

► **Definition 27.** Given  $s \in (!)\Delta_I(X)$ ,  $x \in \mathcal{V}$  and  $\bar{u} \in !\Delta_I(Y)$ , the resource substitution of  $x$  by  $\bar{u}$  in  $s$  is the resource sum  $s\langle \bar{u}/x \rangle \in \mathbb{N}[(!)\Delta_I(X\{Y/x\})]$  defined as follows:

$$\begin{aligned} x\langle \bar{u}/x \rangle &:= \begin{cases} u, & \text{if } \bar{u} = [u], \\ \mathbf{0}_Y, & \text{otherwise,} \end{cases} & (st)\langle \bar{u}/x \rangle &:= \sum_{\bar{u}=\bar{v} \cdot \bar{w}} (s\langle \bar{v}/x \rangle)(\bar{t}\langle \bar{w}/x \rangle), \\ y\langle \bar{u}/x \rangle &:= \begin{cases} y, & \text{if } \bar{u} = 1_Y, \\ \mathbf{0}_{\{y\}}, & \text{otherwise,} \end{cases} & 1_X\langle \bar{u}/x \rangle &:= \begin{cases} 1_{X\{Y/x\}}, & \text{if } \bar{u} = 1_Y, \\ \mathbf{0}_{X\{Y/x\}}, & \text{otherwise,} \end{cases} \\ (\lambda y.s)\langle \bar{u}/x \rangle &:= \lambda y.s\langle \bar{u}/x \rangle, & ([s] \cdot \bar{t})\langle \bar{u}/x \rangle &:= \sum_{\bar{u}=\bar{v} \cdot \bar{w}} [s\langle \bar{v}/x \rangle] \cdot (\bar{t}\langle \bar{w}/x \rangle). \end{aligned}$$

with  $x \neq y$  and in the abstraction case  $y \notin Y$ . We extend it to sums in  $\mathbb{N}[(!)\Delta_I(X)]$  by setting:

$$\begin{aligned} \mathbf{0}_X\langle \bar{u}/x \rangle &:= \mathbf{0}_{X\{Y/x\}} & (s + \mathbf{s})\langle \bar{u}/x \rangle &:= s\langle \bar{u}/x \rangle + \mathbf{s}\langle \bar{u}/x \rangle \\ s\langle \mathbf{0}_Y/x \rangle &:= \mathbf{0}_{X\{Y/x\}} & s\langle (\bar{u} + \bar{\mathbf{u}})/x \rangle &:= s\langle \bar{u}/x \rangle + s\langle \bar{\mathbf{u}}/x \rangle. \end{aligned}$$

It is easy to verify that the definition above does indeed define a resource sum in  $\mathbb{N}[(!)\Delta_I(X\{Y/x\})]$ , and that it is stable under the quotients of Definition 25.

► **Definition 28.** The linear degree  $\deg_x(s)$  of  $x \in \mathcal{V}$  in some  $s \in (!)\Delta_I$  is defined by:

$$\begin{aligned} \deg_x(x) &:= 1, & \deg_x(\lambda y.s) &:= \deg_x(s), \text{ wlog. } x \neq y, & \deg_x(1_X) &:= 0, \\ \deg_x(y) &:= 0, & \deg_x(st) &:= \deg_x(s) + \deg_x(\bar{t}), & \deg_x([s] \cdot \bar{t}) &:= \deg_x(s) + \deg_x(\bar{t}). \end{aligned}$$

The next lemma is not strictly needed, but helps understanding resource substitution.

► **Lemma 29.** Consider  $s \in (!)\Delta_I(X)$ ,  $x \in \mathcal{V}$  and  $\bar{u} \in !\Delta_I(Y)$ , and write  $n := \deg_x(s)$ . Then

$$s\langle \bar{u}/x \rangle = \begin{cases} \sum_{\sigma \in \mathfrak{S}(n)} s[u_{\sigma(1)}/x^{(1)}, \dots, u_{\sigma(n)}/x^{(n)}]\{Y/x\}, & \text{if the cardinality of } \bar{u} \text{ is } n, \\ \mathbf{0}_{X\{Y/x\}}, & \text{otherwise,} \end{cases}$$

where:  $\mathfrak{S}(n)$  is the set of permutations of  $\{1, \dots, n\}$ ;  $u_1, \dots, u_n$  is any enumeration of the elements in  $\bar{u}$ ;  $x^{(1)}, \dots, x^{(n)}$  enumerate the occurrences of  $x$  in  $s$ ; and  $s[u_{\sigma(1)}/x^{(1)}, \dots, u_{\sigma(n)}/x^{(n)}]$  is the  $\lambda$ I-resource term obtained by substituting each element  $u_{\sigma(i)}$  for the occurrence  $x^{(i)}$ .

**Proof.** Straightforward induction on  $s$ . ◀

The next Substitution Lemma concerns the commutation of resource substitutions, and is the analogous of [4, Lemma 2.1.16]. Notice that the assumptions on the substituted variables are stronger than in the usual resource calculus [20, Lemma 2], where only  $x \notin Y$  is required.

► **Lemma 30.** Given  $s \in (!)\Delta_I(X)$ ,  $\bar{u} \in !\Delta_I(Y)$ ,  $\bar{v} \in !\Delta_I(Z)$ , and  $x \in X - Z$ ,  $y \in X \cup Y$ ,

$$s\langle \bar{u}/x \rangle \langle \bar{v}/y \rangle = \sum_{\bar{v}=\bar{v}' \cdot \bar{v}''} s\langle \bar{v}'/y \rangle \langle \bar{u}\langle \bar{v}''/y \rangle/x \rangle.$$

**Proof.** By structural induction on  $s$ , using Lemma 29 in order to apply the induction hypothesis only to those subterms where the hypotheses are actually satisfied. ◀

### 4.3 The operational semantics

We finally endow resource expressions and resource sums with a notion of reduction  $\rightarrow_r$ .

► **Definition 31.** (i) For each  $X \subseteq_f \mathcal{V}$ , define the resource reduction as a relation between  $\lambda$ -resource terms and resource sums, i.e.  $\rightarrow_r \subseteq (!)\Delta_1(X) \times \mathbb{N}[(!)\Delta_1(X)]$ :

$$\frac{}{(\lambda x.s)\bar{t} \rightarrow_r s \langle \bar{t}/x \rangle} \quad \frac{s \rightarrow_r s'}{\lambda x.s \rightarrow_r \lambda x.s'} \quad \frac{s \rightarrow_r s'}{s\bar{t} \rightarrow_r s'\bar{t}} \quad \frac{\bar{t} \rightarrow_r \bar{t}'}{s\bar{t} \rightarrow_r s\bar{t}'} \quad \frac{s \rightarrow_r s'}{[s] \cdot \bar{t} \rightarrow_r [s'] \cdot \bar{t}}$$

- (ii) We denote by  $\rightarrow_r^?$  the reflexive closure of  $\rightarrow_r$  and by  $\twoheadrightarrow_r$  its reflexive-transitive closure.
- (iii) We extend the reduction relation  $\rightarrow_r$  to resource sums  $\mathbb{N}[(!)\Delta_1(X)] \times \mathbb{N}[(!)\Delta_1(X)]$  by saying that  $s \rightarrow_r s'$  and  $\mathbf{t} \rightarrow_r^? \mathbf{t}'$  entail  $s + \mathbf{t} \rightarrow_r s' + \mathbf{t}'$ .

Observe that  $\rightarrow_r$  is well-defined because  $\text{fv}((\lambda x.s)\bar{t}) = \text{fv}(s \langle \bar{t}/x \rangle)$ . Indeed  $\text{fv}((\lambda x.s)\bar{t}) = \text{fv}(s) - \{x\} \cup \text{fv}(\bar{t})$  with  $x \in \text{fv}(s)$ , whence  $\text{fv}(s \langle \bar{t}/x \rangle) = \text{fv}(s) \setminus \{x\} \cup \text{fv}(\bar{t}) = \text{fv}(s) - \{x\} \cup \text{fv}(\bar{t})$ .

► **Example 32.** We use the resource  $\lambda$ -terms  $l$  and  $D_n$  from Example 24.

- (i)  $D_0[z] \rightarrow_r (x1_{\{x\}})\langle [z]/x \rangle = (x\langle [z]/x \rangle)(1_{\{z\}}) + (x\langle 1_{\{z\}}/x \rangle)(1_{\{x\}}\langle [z]/x \rangle) = z1_{\{z\}} + \mathbf{0}_{\{z\}} = z1_{\{z\}}$ . Similarly,  $D_0[l] \rightarrow_r l1_{\emptyset} \rightarrow \mathbf{0}_{\emptyset}$ , while  $D_1[l, l] \rightarrow_r l[l] \rightarrow_r l$  is a non-zero reduction.
- (ii)  $(\lambda x.D_0[x])[z]$  has two redexes. Contracting the outermost first gives  $(\lambda x.D_0[x])[z] \rightarrow_r D_0[z] \rightarrow_r z1_{\{z\}}$ . Contracting the innermost  $(\lambda x.D_0[x])[z] \rightarrow_r (\lambda x.x1_{\{x\}})[z] \rightarrow_r z1_{\{z\}}$ .
- (iii)  $D_1[l, l] \rightarrow_r l[l] + l[l] \rightarrow_r l + l[l] \rightarrow_r l + l = 2.l$ . Thus, sums can arise from single terms.

We show that  $\rightarrow_r$  enjoys the properties of strong confluence and strong normalisation.

► **Theorem 33.** (i) The reduction  $\rightarrow_r$  is strongly normalising.

- (ii)  $\rightarrow_r$  is strongly confluent in the following sense: for all  $s, \mathbf{t}_1, \mathbf{t}_2 \in \mathbb{N}[(!)\Delta_1(X)]$  such that  $s \rightarrow_r \mathbf{t}_1$  and  $s \rightarrow_r \mathbf{t}_2$ , there exists  $\mathbf{u} \in \mathbb{N}[(!)\Delta_1(X)]$  such that  $\mathbf{t}_1 \rightarrow_r^? \mathbf{u}$  and  $\mathbf{t}_2 \rightarrow_r^? \mathbf{u}$ .

**Proof.** (i) The size  $\text{size}(s) \in \mathbb{N}$  of a resource expression  $s \in (!)\Delta_1$  is defined by structural induction as usual, with base cases  $\text{size}(x) = 1$  and  $\text{size}(1_X) = 0$ , so that  $\text{size}(1_X \cdot \bar{t}) = \text{size}(\bar{t})$ . The *sum-size* of a resource sum  $\mathbf{s}$  is the finite multiset defined by  $\text{ssize}(\mathbf{0}_X) = []$  and  $\text{ssize}(s + \mathbf{t}) = [\text{size}(s)] \cdot \text{ssize}(\mathbf{t})$ . Sum-sizes are well-ordered by the usual well-founded ordering  $\prec_{\mathbb{N}}$  on finite multisets over  $\mathbb{N}$  (see [41, §A.6]).

Now, assume that  $s \rightarrow_r s'$ . Since the contraction of a redex suppresses an abstraction and there is no duplication, we get  $\text{ssize}(s) \prec_{\mathbb{N}} \text{ssize}(s')$ . Conclude since  $\prec_{\mathbb{N}}$  is well-founded.

- (ii) The sequence of lemmas leading to the result follows a well-trodden path in the resource calculus [44, 10]. It is sufficient to check that they generalize to our setting. We give some details in the appendix. ◀

By (ii) above every  $s \in \mathbb{N}[(!)\Delta_1(X)]$  has a unique  $r$ -normal form which is denoted  $\text{nf}(s)$ .

## 5 Taylor approximation for memory trees

In its original formulation, Ehrhard and Regnier's Taylor expansion translates a  $\lambda$ -term as a power series of iterated derivatives [19]. We now introduce a *qualitative* Taylor expansion [32] specifically designed for the  $\lambda$ -calculus, having as target the  $\lambda$ -resource calculus.

### 5.1 The Taylor approximation

Intuitively, a qualitative Taylor expansion should associate each term  $M \in \Delta_1(X)$  with a *set* of resource approximants  $\mathcal{T}_m(M) \subseteq \Delta_1(X)$ . Therefore the codomain of  $\mathcal{T}_m(-)$  should be  $\coprod_{X \subseteq_f \mathcal{V}} 2^{\Delta_1(X)}$ , and what we actually define is  $\mathcal{T}_m(M) := (X, \mathcal{T}_m^\bullet(M))$  with  $\mathcal{T}_m^\bullet(M) \subseteq \Delta_1(X)$ .

Notice that, whereas all previous constructions of disjoint unions  $\coprod_{X \subseteq_f \mathcal{V}}$  were formed from genuinely disjoint sets, this is no longer the case here, as  $\emptyset \in 2^{\Delta_1(X)}$ , for all  $X$ .

- **Notation 34.** (i) For  $X \subseteq_f \mathcal{V}$  and  $\mathcal{X} \in 2^{\Delta_1(X)}$ , we write  $\text{fv}(X, \mathcal{X}) := X$ .  
(ii) We let  $\subseteq^\bullet$  denote the order relation on  $\coprod_{X \subseteq_f \mathcal{V}} 2^{\Delta_1(X)}$  such that  $(X, \mathcal{X}) \subseteq^\bullet (Y, \mathcal{Y})$  whenever  $X = Y$  and  $\mathcal{X} \subseteq \mathcal{Y}$ . We write  $\cup^\bullet$  for the corresponding least upper bounds.  
(iii) We write  $s \in^\bullet (X, \mathcal{X})$  to mean that  $s \in \mathcal{X}$ .  
(iv) Given  $\mathcal{X} \in 2^{\Delta_1(X)}$ ,  $\mathcal{Y} \in 2^{\Delta_1(Y)} - \{\emptyset\}$  and  $x \in X$ , we write:

$$\lambda x.(X, \mathcal{X}) := (X - \{x\}, \{\lambda x.s \mid s \in \mathcal{X}\}), \quad (X, \mathcal{X})\mathcal{Y} := (X \cup Y, \{s\bar{t} \mid s \in \mathcal{X}, \bar{t} \in \mathcal{Y}\}).$$

- **Definition 35.** (i) The Taylor expansion  $\mathcal{T}_m(M) \in \coprod_{X \subseteq_f \mathcal{V}} 2^{\Delta_1(X)}$  of a  $\lambda$ I-term  $M$ , is defined together with  $\mathcal{T}_m^1(M) \subset \Delta_1(\text{fv}(M))$  by mutual induction:

$$\begin{aligned} \mathcal{T}_m(x) &:= (\{x\}, \{x\}), \quad \mathcal{T}_m(\lambda x.M) := \lambda x.\mathcal{T}_m(M), \quad \mathcal{T}_m(MN) := \mathcal{T}_m(M)\mathcal{T}_m^1(N), \\ \mathcal{T}_m^1(M) &:= \{1_{\text{fv}(M)}\} \cup \{[t_0, \dots, t_n] \mid n \in \mathbb{N}, t_0, \dots, t_n \in^\bullet \mathcal{T}_m(M)\}. \end{aligned}$$

- (ii) The above definition is extended to memory trees by setting, for all  $M \in \Lambda_I$ ,

$$\mathcal{T}_m(\text{MT}(M)) := \bigcup_{A \in \text{App}_m(M)}^\bullet \mathcal{T}_m(A), \quad \text{together with } \begin{cases} \mathcal{T}_m(\perp_X) := (X, \emptyset), \\ \mathcal{T}_m^1(\perp_X) := \{1_X\}. \end{cases}$$

► **Remark 36.** For all  $M$ , we have  $\text{fv}(\mathcal{T}_m(M)) = \text{fv}(M)$ . Similarly,  $\text{fv}(\mathcal{T}_m(\text{MT}(M))) = \text{fv}(M)$ , due to Remark 17 and the way we ordered  $\coprod_{X \subseteq_f \mathcal{V}} 2^{\Delta_1(X)}$ .

- **Example 37.** (i)  $\mathcal{T}_m(I) = (\emptyset, \{I\})$ ,  $\mathcal{T}_m(D) = (\emptyset, \{D_n \mid n \in \mathbb{N}\})$ .  
(ii)  $\mathcal{T}_m(\text{MT}(Yf)) = (\{f\}, \mathcal{X}_{Yf})$  and  $\mathcal{T}_m(\text{MT}(\mathbf{I}_l f)) = (\{f, l\}, \mathcal{X}_{\mathbf{I}_l f})$ , where the sets of approximants can be described as the smallest (in fact, unique) subsets of  $\Delta_1$  such that:

$$\begin{aligned} \mathcal{X}_{Yf} &= \{f1_{\{f\}}\} \cup \{f[t_0, \dots, t_n] \mid n \in \mathbb{N}, t_0, \dots, t_n \in \mathcal{X}_{Yf}\}, \\ \mathcal{X}_{\mathbf{I}_l f} &= \{f1_{\{f, l\}}\} \cup \{f[t_0, \dots, t_n] \mid n \in \mathbb{N}, t_0, \dots, t_n \in \mathcal{X}_{\mathbf{I}_l f}\}. \end{aligned}$$

We now describe how resource reduction acts on Taylor expansions, *i.e.* on potentially infinite sets of resource expressions.

- **Notation 38.** (i) Given  $\mathbf{s} \in \mathbb{N}^{(!)\Delta_1(X)}$ , we denote by  $|\mathbf{s}| \in \coprod_{X \subseteq_f \mathcal{V}} 2^{\Delta_1(X)}$  its *support*, defined by  $|\mathbf{0}_X| := (X, \emptyset)$  and  $|\mathbf{s} + \mathbf{t}| := (X, \{s\}) \cup^\bullet |\mathbf{t}|$ . Notice that  $s \in^\bullet |\mathbf{t}|$  whenever  $s$  bears a non-zero coefficient in  $\mathbf{t}$  (*i.e.*  $\mathbf{t} = \mathbf{s} + \mathbf{t}'$ , for some  $\mathbf{t}'$ ).  
(ii) Given  $\mathcal{X} \in 2^{(!)\Delta_1(X)}$ , we write  $\text{nf}(X, \mathcal{X}) := \bigcup_{s \in \mathcal{X}}^\bullet |\text{nf}(s)|$ . Notice that  $\text{fv}(\text{nf}(X, \mathcal{X})) = X$ , and that  $\text{nf}(X, \mathcal{X}) = (X, \emptyset)$  if and only if  $s \twoheadrightarrow_r \mathbf{0}_X$ , for all  $s \in \mathcal{X}$ .

This allows us to state the following theorem, adapting [19, Theorem 2].

- **Theorem 39 (commutation).** For all  $M \in \Lambda_I$ ,  $\text{nf}(\mathcal{T}_m(M)) = \mathcal{T}_m(\text{MT}(M))$ .

We outline a proof in the following sequence of lemmas, similar to the one in [2]; alternatively, the variants of [19, 20], [36] and [10, 11] could also be adapted.

- **Lemma 40.** For all resource sums  $\mathbf{s}$  such that  $|\mathbf{s}| \subseteq^\bullet \mathcal{T}_m(M)$ , there exists a reduction  $M \twoheadrightarrow_\beta N$  such that  $|\text{nf}(\mathbf{s})| \subseteq^\bullet \mathcal{T}_m(N)$ .

**Proof.** By induction on the length of the longest reduction  $\mathbf{s} \twoheadrightarrow_r \text{nf}(\mathbf{s})$ . If it is 0, take  $N := M$ . Otherwise, the longest reduction is  $\mathbf{s} = t + \mathbf{u} \rightarrow_r \mathbf{t}' + \mathbf{u} \twoheadrightarrow_r \text{nf}(\mathbf{s})$ . By firing all redexes in  $M$  and  $\mathbf{u}$  at the same position as the redex fired in  $t \rightarrow_r \mathbf{t}'$  (formally we do this by induction on this reduction), we obtain  $M \rightarrow_\beta M'$  and  $\mathbf{u} \twoheadrightarrow_r \mathbf{u}'$  such that  $|\mathbf{t}' + \mathbf{u}'| \subseteq^\bullet \mathcal{T}_m(M')$ . By confluence,  $\mathbf{t}' + \mathbf{u}' \twoheadrightarrow_r \text{nf}(\mathbf{s})$ . We conclude by induction on this (shorter) reduction. ◀

► **Lemma 41.** *If  $t \in^\bullet \mathcal{T}_m(N)$  is in r-nf, then there exists  $A \in \text{App}_m(N)$  such that  $t \in^\bullet \mathcal{T}_m(A)$ .*

**Proof.** By induction on  $t$ . A r-nf is also a hnf, hence  $t = \lambda \vec{x}. y \bar{t}_1 \cdots \bar{t}_k$  and  $N = \lambda \vec{x}. y N_1 \cdots N_k$ . For  $1 \leq i \leq k$ ,  $\bar{t}_i \in \mathcal{T}_m^1(N_i)$ . If  $\bar{t}_i = 1_{\text{fv}(N_i)}$  define  $A_i := \perp_{\text{fv}(N_i)}$ . If  $\bar{t}_i = [t_i^1, \dots, t_i^n]$ , each  $t_i^j$  is in r-nf, by induction there is  $A_i^j \in \text{App}_m(N_i)$ ,  $t_i^j \in^\bullet \mathcal{T}_m(A_i^j)$ . Define  $A_i := \bigsqcup_{j=1}^n A_i^j \in \text{App}_m(N_i)$ . Finally,  $A := \lambda \vec{x}. y A_1 \cdots A_k \in \text{App}_m(N)$  and  $t \in^\bullet \mathcal{T}_m(A)$ . ◀

► **Lemma 42** (monotonicity of  $\mathcal{T}_m(-)$ ). (i)  $\mathcal{T}_m(A) \subseteq^\bullet \mathcal{T}_m(A')$  if and only if  $A \sqsubseteq A'$ .  
(ii) For all  $N \in \Lambda_1$ , we have  $\mathcal{T}_m(\omega_m(N)) \subseteq^\bullet \mathcal{T}_m(N)$ .

**Proof.** Immediate inductions (i) on  $A$  and  $A'$ , and (ii) on the head structure of  $N$ . ◀

► **Lemma 43** (simulation of  $\rightarrow_\beta$ ). *If  $M \rightarrow_\beta N$ , then  $\mathcal{T}_m(N) = \bigcup_{s \in^\bullet \mathcal{T}_m(M)} |\mathbf{t}_s|$  for resource sums  $\mathbf{t}_s \in \mathbb{N}[\Delta_1(\text{fv}(M))]$  such that  $\forall s \in^\bullet \mathcal{T}_m(M), s \twoheadrightarrow_r \mathbf{t}_s$ .*

**Proof.** One first shows by induction on  $P$  that for all  $P, Q \in \Lambda_1$  and  $x \in \text{fv}(P)$ ,  $\mathcal{T}_m(P[Q/x]) = \bigcup_{s \in^\bullet \mathcal{T}_m(P)} \bigcup_{t \in \mathcal{T}_m^1(Q)} |s \langle \bar{t}/x \rangle|$ . Then the proof is an induction on  $M \rightarrow_\beta N$ , using the previous equality in the base case. See the details of a similar proof in [11, Lemmas 4.1 and 4.2]. ◀

► **Lemma 44.** *If  $A \in \mathcal{A}_m$  then all  $s \in^\bullet \mathcal{T}_m(A)$  are in r-nf.*

**Proof.** Immediate induction on  $A$ . ◀

Everything is now in place to prove the Commutation Theorem 39.

**Proof of Theorem 39.** By Remark 36,  $\text{fv}(\text{nf}(\mathcal{T}_m(M))) = \text{fv}(M) = \text{fv}(\mathcal{T}_m(\text{MT}(M)))$ . Thus, it is sufficient to prove that, for  $t \in \Delta_1(\text{fv}(M))$ :  $t \in^\bullet \text{nf}(\mathcal{T}_m(M))$  if and only if  $t \in^\bullet \mathcal{T}_m(\text{MT}(M))$ .

Take  $t \in^\bullet \text{nf}(\mathcal{T}_m(M))$ , i.e.  $t \in |\text{nf}(s)|$  for some  $s \in^\bullet \mathcal{T}_m(M)$ . Then, by Lemma 40, there exists a reduction  $M \twoheadrightarrow_\beta N$  such that  $|\text{nf}(s)| \subseteq^\bullet \mathcal{T}_m(N)$ . Hence  $t \in^\bullet \mathcal{T}_m(N)$  and  $t$  is in r-nf: Lemma 41 ensures that  $t \in^\bullet \mathcal{T}_m(A)$  for some  $A \in \text{App}_m(N) = \text{App}_m(M)$  (by Proposition 20). This means exactly that  $t \in^\bullet \mathcal{T}_m(\text{MT}(M))$ .

Conversely, take  $t \in^\bullet \mathcal{T}_m(\text{MT}(M))$ , i.e.  $t \in^\bullet \mathcal{T}_m(A)$  for some  $A \in \text{App}_m(M)$ . By definition there is a reduction  $M \twoheadrightarrow_\beta N$  such that  $A \sqsubseteq \omega_m(N)$ . By Lemma 42,  $t \in^\bullet \mathcal{T}_m(A) \subseteq^\bullet \mathcal{T}_m(\omega_m(N)) \subseteq^\bullet \mathcal{T}_m(N)$ . By iterated applications of Lemma 43, there is an  $s \in^\bullet \mathcal{T}_m(M)$  such that  $s \twoheadrightarrow_r \mathbf{t}_s$  and  $t \in |\mathbf{t}_s|$ . By Lemma 44,  $t$  is in r-nf, therefore  $t \in^\bullet |\text{nf}(s)| \subseteq^\bullet \text{nf}(\mathcal{T}_m(M))$ . ◀

## 5.2 The $\lambda\mathbf{I}$ -theory of memory trees

Consider the equivalence  $\mathcal{M}$  on  $\Lambda_1$ , defined by  $M =_{\mathcal{M}} N$  if and only if  $\text{MT}(M) = \text{MT}(N)$ . Thanks to Theorem 39, we are now able to show that this equivalence is a  $\lambda\mathbf{I}$ -theory.

► **Corollary 45.** *For  $M, N \in \Lambda_1$ ,  $M =_{\mathcal{M}} N$  if and only if  $\text{nf}(\mathcal{T}_m(M)) = \text{nf}(\mathcal{T}_m(N))$ .*

**Proof.** ( $\Rightarrow$ ) Immediate by Theorem 39. ( $\Leftarrow$ ) Take  $M, N$  such that  $\text{nf}(\mathcal{T}_m(M)) = \text{nf}(\mathcal{T}_m(N))$ . By Theorem 21, it is sufficient to prove  $\text{App}_m(M) = \text{App}_m(N)$ . Take  $A \in \text{App}_m(M)$ . If  $A = \perp_X$ , then  $A \in \text{App}_m(N)$  is trivial since  $\text{App}_m(N)$  is downward closed. Otherwise, by hypothesis and Theorem 39,  $\mathcal{T}_m(A) \subseteq^\bullet \bigcup_{B \in \text{App}_m(N)} \mathcal{T}_m(B)$ . There is a  $B \in \text{App}_m(N)$  such that  $\llbracket A \rrbracket_r \in^\bullet \mathcal{T}_m(B)$ , where  $\llbracket A \rrbracket_r \in^\bullet \mathcal{T}_m(A)$  is defined by

$$\begin{aligned} \llbracket x \rrbracket_r &:= x, \\ \llbracket \lambda \vec{x}. y A_1 \cdots A_k \rrbracket_r &:= \lambda \vec{x}. y \llbracket A_1 \rrbracket_r^! \cdots \llbracket A_k \rrbracket_r^!, \end{aligned} \quad \llbracket A \rrbracket_r^! := \begin{cases} 1_X, & \text{if } A = \perp_X, \\ \llbracket \llbracket A \rrbracket_r \rrbracket, & \text{otherwise.} \end{cases}$$

By induction on  $A$ , one shows that  $\llbracket A \rrbracket_r \in^\bullet \mathcal{T}_m(B)$  implies  $A \sqsubseteq B$ , hence  $A \in \text{App}_m(N)$  by downward-closure (Lemma 19(ii)). This shows that  $\text{App}_m(M) \subseteq \text{App}_m(N)$ . The converse inclusion is symmetric. ◀

► **Corollary 46.**  $\mathcal{M}$  is a  $\lambda\mathbf{l}$ -theory.

**Proof.** The relation  $=_{\mathcal{M}}$  is clearly an equivalence. By Proposition 20 and Theorem 21, for all  $M, N \in \Lambda_{\mathbf{l}}$ ,  $M =_{\beta} N$  entails  $M =_{\mathcal{M}} N$ .

For compatibility with abstraction, take  $M =_{\mathcal{M}} N$  and  $x \in \text{fv}(M) \cap \text{fv}(N)$ . By Corollary 45, we get  $\text{nf}(\mathcal{T}_{\mathbf{m}}(\lambda x.M)) = \lambda x.\text{nf}(\mathcal{T}_{\mathbf{m}}(M)) = \lambda x.\text{nf}(\mathcal{T}_{\mathbf{m}}(N)) = \text{nf}(\mathcal{T}_{\mathbf{m}}(\lambda x.N))$ .

For compatibility with application, by Theorem 33 observe that for all  $M, N \in \Lambda_{\mathbf{l}}$ ,  $\text{nf}(\mathcal{T}_{\mathbf{m}}(MN)) = \text{nf}(\mathcal{T}_{\mathbf{m}}(M)\mathcal{T}_{\mathbf{m}}^{\dagger}(N)) = \text{nf}(\text{nf}(\mathcal{T}_{\mathbf{m}}(M))\text{nf}(\mathcal{T}_{\mathbf{m}}^{\dagger}(N)))$ , where  $\text{nf}(\mathcal{T}_{\mathbf{m}}^{\dagger}(N))$  is defined by adapting Notation 38 to sets of resource bags. Therefore,  $\text{nf}(\mathcal{T}_{\mathbf{m}}(M)) = \text{nf}(\mathcal{T}_{\mathbf{m}}(M'))$  and  $\text{nf}(\mathcal{T}_{\mathbf{m}}(N)) = \text{nf}(\mathcal{T}_{\mathbf{m}}(N'))$  imply  $\text{nf}(\mathcal{T}_{\mathbf{m}}(MN)) = \text{nf}(\mathcal{T}_{\mathbf{m}}(M'N'))$ , and we can conclude the proof by Corollary 45. ◀

## 6 Conclusions and future work

In this paper we introduced the memory trees for the  $\lambda\mathbf{l}$ -calculus, together with two theories of program approximations: the former based on finite trees, the latter on resource approximants and Taylor expansion. Our pioneering results look encouraging, so we believe that this approach deserves further investigations.

### 6.1 Further work on the $\lambda\mathbf{l}$ -calculus

The memory trees introduced in Definition 10 are an adaptation of Böhm trees, since they regard as meaningless the terms without hnf. By considering as meaningless the subset of zero-terms<sup>1</sup> one obtains Lévy-Longo trees [29, 30], and by taking mute terms<sup>2</sup> as meaningless one obtains Berarducci trees [7]. Our definition of memory trees can be readily adapted to both settings by appropriately modifying the notion of meaningless. Preliminary investigations indicate that all our results extend seamlessly to the Lévy-Longo version. Regarding the Berarducci version, the direct approximants can be generalized without any issues (except for the fact that an oracle is needed, see [43]). We are currently studying whether the corresponding  $\lambda\mathbf{l}$ -resource calculus and Taylor expansion could also be designed.

Our investigation of memory trees started from semantical considerations. The relational model with infinite multiplicities  $\mathcal{E}$ , defined in [9], distinguishes  $\Omega$  from  $\Omega x$ , and  $\Upsilon$  from  $\mathbf{t}_l$ , just like our memory trees. Unlike our memory trees, it equates  $\lambda x.x(\Omega y)(\Omega z)$  and  $\lambda x.x(\Omega z)(\Omega y)$ , thus the induced theory is different from memory trees equality ( $=_{\mathcal{M}}$ ).

Finding a denotational model whose theory captures exactly the memory tree equality would reinforce our belief that they are a natural notion of trees for the  $\lambda\mathbf{l}$ -calculus.

► **Problem 1.** *Is there a denotational model of  $\Lambda_{\mathbf{l}}$  whose theory is exactly  $\mathcal{M}$ ?*

We believe that finding a denotational model and studying its categorical properties are important steps towards a deeper mathematical understanding of our Taylor expansion. Currently, it is unclear whether our resource calculus stands on a solid notion of differentiation, as it is the case for the usual resource calculus [18], or if it is an *ad hoc* adaptation.

► **Problem 2.** *Is the  $\lambda\mathbf{l}$ -resource calculus with memory representing some notion of derivative?*

We now discuss more speculative extensions, going beyond the setting of  $\lambda\mathbf{l}$ -calculus.

<sup>1</sup> I.e., terms without an hnf that never reduce to an abstraction.

<sup>2</sup> Also called *root active*, mute terms have the property that all their reducts can reduce to a redex.

## 6.2 What about the full $\lambda$ -calculus?

We have seen that the model  $\mathcal{E}$  from [9] distinguishes some meaningless terms having different free variables, but equates some  $\lambda$ I-terms having different memory trees. Characterizing its theory seems a very natural question that has been completely overlooked in the literature.

► **Problem 3.** *Is there an operational characterization of the  $\lambda$ -theory induced by  $\mathcal{E}$ ?*

The fact that  $\mathcal{E}$  is a model of the full  $\lambda$ -calculus witnesses the existence of consistent  $\lambda$ -theories that remember variables that are pushed into infinity in the Böhm tree semantics. It is then natural to wonder whether it makes sense to extend the definition of memory trees to arbitrary  $\lambda$ -terms. The idea is to consider the set of *permanent free variables* of  $M \in \Lambda$ :

$$\text{pfv}(M) = \{x \mid \forall N \in \Lambda. M \rightarrow_{\beta} N \text{ implies } x \in \text{fv}(N)\}$$

Intuitively,  $\text{pfv}(M)$  is the set of free variables of  $M$  that are never erased along a reduction.

► **Definition 47.** *The memory tree of a  $\lambda$ -term  $M$  can be then defined by setting:*

- $\text{MT}(M) = \lambda \vec{x}. y. \text{pfv}(M_1) \text{ MT}(M_1) \dots \text{pfv}(M_k) \text{ MT}(M_k)$ , if  $M \rightarrow_h \lambda x_1 \dots x_n. y. M_1 \dots M_k$ ;
- $\perp_{\text{pfv}(M)}$ , otherwise.

For  $M \in \Lambda_1$  the definition above is equivalent to Definition 10, since  $\text{pfv}(M) = \text{fv}(M)$ . For an arbitrary  $M \in \Lambda$ , one needs an oracle to compute  $\text{pfv}(M)$ , but an oracle is already needed to determine whether  $M$  has an hnf, therefore the logical complexity does not increase.

Preliminary investigations show that the above notion of memory tree remains invariant by  $\rightarrow_{\beta}$ . The key property which is exploited to prove this result is that  $\text{pfv}(\lambda \vec{x}. y. M_1 \dots M_k) = \{y\} \cup \text{pfv}(M_1) \cup \dots \cup \text{pfv}(M_k)$ . The main problem of Definition 47 is that the equality induced on  $\lambda$ -term is not a  $\lambda$ -theory because it is not compatible with application. The following counterexample exploits the fact that, in general,  $\text{pfv}(MN) \neq \text{pfv}(M) \cup \text{pfv}(N)$ . Define:

$$W = \lambda xy. yxy, \quad M = \lambda z. W(zv_1v_2)W, \quad N = \lambda z. W(zv_2v_1)W, \quad \text{for } v_1, v_2 \in \mathcal{V}.$$

We have  $M \rightarrow_h \lambda z. (\lambda y. y(zv_1v_2)y)W \rightarrow_h M$  and  $N \rightarrow_h \lambda z. (\lambda y. y(zv_2v_1)y)W \rightarrow_h N$ , whence  $\text{MT}(M) = \text{MT}(N) = \perp_{\{v_1, v_2\}}$ . By applying the 1st projection  $K$ , we obtain  $MK \rightarrow_{\beta} Wv_1W$  and  $NK \rightarrow_{\beta} Wv_2W$ , therefore  $\text{MT}(MK) = \perp_{\{v_1\}} \neq \perp_{\{v_2\}} = \text{MT}(NK)$ . We conclude that the equality induced on  $\Lambda$  is not a  $\lambda$ -theory, because it is not compositional. This counterexample is particularly strong because it shows that this is independent from the fact that memory trees are generalizations of Böhm trees and not, say, Berarducci trees. Indeed, the  $M, N$  constructed above have the same Böhm tree, Lévy-Longo tree and Berarducci tree.

A natural question is whether memory trees can be used as a meaningful notion of observation, in the sense of Morris's observational equivalences [35].

► **Problem 4.** *Is there a denotational model inducing the following  $\lambda$ -theory?*

$$M \equiv N \iff \forall C[] . \text{MT}(C[M]) = \text{MT}(C[N])$$

where  $C[]$  denotes a  $\lambda$ -calculus context (namely, a  $\lambda$ -term containing a hole  $[]$ ), and  $C[M]$  the  $\lambda$ -term obtained by replacing  $M$  for the hole  $[]$  in  $C[]$ , possibly with capture of free variables.

---

## References

- 1 Sandra Alves and Mário Florido. Structural rules and algebraic properties of intersection types. In Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu, editors, *Theoretical Aspects of Computing - ICTAC 2022 - 19th International Colloquium, Tbilisi, Georgia, September 27-29, 2022, Proceedings*, volume 13572 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2022. doi:10.1007/978-3-031-17715-6\_6.

- 2 Davide Barbarossa and Giulio Manzonetto. Taylor subsumes Scott, Berry, Kahn and Plotkin. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–23, 2019. doi:10.1145/3371069.
- 3 Henk Barendregt. The type free lambda calculus. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 1091–1132. North-Holland, Amsterdam, 1977.
- 4 Henk Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.
- 5 Henk Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications, 2022. URL: <https://www.collegepublications.co.uk/logic/mlf/?00035>.
- 6 Stefano Berardi and Ugo de'Liguoro. Non-monotonic pre-fix points and learning. *Fundam. Informaticae*, 150(3-4):259–280, 2017. doi:10.3233/FI-2017-1470.
- 7 Alessandro Berarducci. Infinite  $\lambda$ -calculus and non-sensible models. In Marcel Dekker, editor, *Logic and Algebra*, volume 180, pages 339–377, New York, 1996. Presented to the conference in honour of Roberto Magari, Pontignano 1994.
- 8 Flavien Breuvar, Giulio Manzonetto, and Domenico Ruoppolo. Relational graph models at work. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:2)2018.
- 9 Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. Exponentials with infinite multiplicities. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 170–184. Springer, 2010. doi:10.1007/978-3-642-15205-4\_16.
- 10 Rémy Cerda. *Taylor Approximation and Infinitary  $\lambda$ -Calculi*. Phd thesis, Aix-Marseille Université, 2024. URL: <https://hal.science/tel-04664728>.
- 11 Rémy Cerda and Lionel Vaux Auclair. Finitary simulation of infinitary  $\beta$ -reduction via Taylor expansion, and applications. *Logical Methods in Computer Science*, 19, 2023. doi:10.46298/lmcs-19(4:34)2023.
- 12 A. Church. *The calculi of lambda conversion*. Princeton University Press, 1941.
- 13 Lukasz Czajka. A new coinductive confluence proof for infinitary lambda calculus. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:31)2020.
- 14 Daniel de Carvalho and Lorenzo Tortora de Falco. The relational model is injective for multiplicative exponential linear logic (without weakenings). *Ann. Pure Appl. Log.*, 163(9):1210–1236, 2012. URL: <https://doi.org/10.1016/j.apal.2012.01.004>, doi:10.1016/J.APAL.2012.01.004.
- 15 Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Inf. Comput.*, 248:104–129, 2016. URL: <https://doi.org/10.1016/j.ic.2015.12.010>, doi:10.1016/J.IC.2015.12.010.
- 16 Aloÿs Dufour and Damiano Mazza. Böhm and Taylor for all! In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, volume 299 of *LIPICs*, pages 29:1–29:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: <https://doi.org/10.4230/LIPICs.FSCD.2024.29>, doi:10.4230/LIPICs.FSCD.2024.29.
- 17 Lavinia Egidi, Furio Honsell, and Simona Ronchi Della Rocca. Operational, denotational and logical descriptions: a case study. *Fundam. Informaticae*, 16(1):149–169, 1992.
- 18 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.
- 19 Thomas Ehrhard and Laurent Regnier. Böhm trees, Krivine’s machine and the Taylor expansion of lambda-terms. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers (CiE 2006)*, pages 186–197, 2006. doi:10.1007/11780342\_20.

- 20 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theoretical Computer Science*, 403(2):347–372, 2008. doi:10.1016/j.tcs.2008.06.001.
- 21 Jörg Endrullis, Dimitri Hendriks, and Jan Willem Klop. Highlights in infinitary rewriting and lambda calculus. *Theor. Comput. Sci.*, 464:48–71, 2012. URL: <https://doi.org/10.1016/j.tcs.2012.08.018>, doi:10.1016/J.TCS.2012.08.018.
- 22 Silvia Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Log.*, 37(1):44–52, 1996. URL: <https://doi.org/10.1305/ndjfl/1040067315>, doi:10.1305/NDJFL/1040067315.
- 23 Furio Honsell and Marina Lenisa. Some results on the full abstraction problem for restricted lambda calculi. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 84–104. Springer, 1993. doi:10.1007/3-540-57182-5\_6.
- 24 J. Martin E. Hyland. A syntactic characterization of the equality in some models for the  $\lambda$ -calculus. *Journal London Mathematical Society (2)*, 12(3):361–370, 1975.
- 25 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. doi:10.23638/LMCS-15(1:6)2019.
- 26 Bart Jacobs. Semantics of lambda-I and of other substructure lambda calculi. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 1993. URL: <https://doi.org/10.1007/BFb0037107>, doi:10.1007/BFb0037107.
- 27 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculi and Böhm models. In Jieh Hsiang, editor, *Rewriting Techniques and Applications, 6th International Conference, RTA-95, Kaiserslautern, Germany, April 5-7, 1995, Proceedings*, volume 914 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1995. doi:10.1007/3-540-59200-8\_62.
- 28 J. Ketema, Stefan Blom, Takahito Aoto, and Jakob Grue Simonsen. *Rewriting Transfinite Terms*, pages 129–144. Lulu Press Inc, United States, 2009.
- 29 Jean-Jacques Lévy. An algebraic interpretation of the  $\lambda\beta K$ -calculus; and an application of a labelled  $\lambda$ -calculus. *Theor. Comput. Sci.*, 2(1):97–114, 1976. doi:10.1016/0304-3975(76)90009-8.
- 30 Giuseppe Longo. Set-theoretical models of  $\lambda$ -calculus: theories, expansions, isomorphisms. *Annals of Pure and Applied Logic*, 24(2):153–188, 1983. doi:10.1016/0168-0072(83)90030-1.
- 31 Stefania Lusin and Antonino Salibra. The lattice of lambda theories. *J. Log. Comput.*, 14(3):373–394, 2004. URL: <https://doi.org/10.1093/logcom/14.3.373>, doi:10.1093/LOGCOM/14.3.373.
- 32 Giulio Manzonetto and Michele Pagani. Böhm’s Theorem for resource lambda calculus through Taylor Expansion. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2011. doi:10.1007/978-3-642-21691-6\_14.
- 33 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. URL: <https://doi.org/10.1093/logcom/exz013>, doi:10.1093/LOGCOM/EXZ013.
- 34 Paul-André Melliès. Higher-order parity automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005077.
- 35 J.H. Morris. *Lambda calculus models of programming languages*. Phd thesis, Massachusetts Institute of Technology (MIT), 1968.

- 36 Federico Olimpieri and Lionel Vaux Auclair. On the Taylor expansion of  $\lambda$ -terms and the groupoid structure of their rigid approximants. *Logical Methods in Computer Science*, 18, 2022. doi:10.46298/lmcs-18(1:1)2022.
- 37 Michele Pagani and Paolo Tranquilli. Parallel reduction in resource lambda-calculus. In Zhenjiang Hu, editor, *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*, volume 5904 of *Lecture Notes in Computer Science*, pages 226–242. Springer, 2009. doi:10.1007/978-3-642-10672-9\_17.
- 38 Simona Ronchi Della Rocca. Characterization theorems for a filter lambda model. *Inf. Control.*, 54(3):201–216, 1982. doi:10.1016/S0019-9958(82)80022-3.
- 39 Dana S. Scott. Continuous lattices. In Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer, 1972.
- 40 Paula Severi and Fer-Jan de Vries. The infinitary lambda calculus of the infinite  $\eta$ -Böhm trees. *Math. Struct. Comput. Sci.*, 27(5):681–733, 2017. doi:10.1017/S096012951500033X.
- 41 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
- 42 Steffen van Bakel. Complete restrictions of the intersection type discipline. *Theor. Comput. Sci.*, 102(1):135–163, 1992. doi:10.1016/0304-3975(92)90297-S.
- 43 Steffen van Bakel, Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Fer-Jan de Vries. Intersection types for lambda-trees. *Theor. Comput. Sci.*, 272(1-2):3–40, 2002. doi:10.1016/S0304-3975(00)00346-7.
- 44 Lionel Vaux. Normalizing the Taylor expansion of non-deterministic  $\lambda$ -terms, via parallel reduction of resource vectors. *Logical Methods in Computer Science*, 15, 2019. doi:10.23638/LMCS-15(3:9)2019.
- 45 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. URL: <https://doi.org/10.1137/0205036>.
- 46 Christopher P. Wadsworth. Approximate reduction and lambda calculus models. *SIAM J. Comput.*, 7(3):337–356, 1978. doi:10.1137/0207028.

## A

 Proofs of Section 3

► **Lemma 15** (recalled from Page 7). *There is a bijection between  $\mathcal{A}_m$  and the set of finite memory trees of  $\lambda$ -terms.*

**Proof.** Given  $A \in \mathcal{A}_m$ , define a  $\lambda$ -term  $M_A$  by structural induction on  $A$ :

- if  $A = \perp_{\{x_1, \dots, x_n\}}$  then  $M_A := \Omega x_1 \cdots x_n$ .
- if  $A = \lambda \vec{x}. y A_1 \cdots A_k$  then  $M_A := \lambda \vec{x}. y (M_{A_1}) \cdots (M_{A_k})$ .

Define a map  $\iota$  associating to  $A$  the memory tree  $\text{MT}(M_A)$ , i.e.  $\iota(A) := \text{MT}(M_A)$ .

Conversely, let  $M \in \Lambda_1$  be such that  $T := \text{MT}(M)$  is finite. Since  $T$  is finite, we can construct an approximant  $\iota^-(T)$  by structural induction on  $T$ . There are two cases:

- $T = \perp_X$ , with  $X = \text{fv}(M)$ . In this case, define  $\iota^-(T) := \perp_X$ .
- If  $T = \lambda \vec{x}. y \cdot^{X_1} T_1 \cdots^{X_k} T_k$  then  $M \twoheadrightarrow_h \lambda \vec{x}. y M_1 \cdots M_k$  with  $T_i = \text{MT}(M_i)$  and  $X = \text{fv}(M_i)$ , for all  $i$  ( $1 \leq i \leq k$ ). In this case, define  $\iota^-(T) = \lambda \vec{x}. y (\iota^-(T_1)) \cdots (\iota^-(T_k))$ .

A straightforward induction shows  $\iota(\iota^-(T)) = T$  and  $\iota^-(\iota(A)) = A$ . ◀

► **Lemma 19** (recalled from Page 7).

- (i) *For every  $M, N \in \Lambda_1$ ,  $M \rightarrow_\beta N$  entails  $\omega_m(M) \sqsubseteq \omega_m(N)$ .*
- (ii) *For all  $M \in \Lambda_1(X)$ ,  $\text{App}_m(M)$  is an ideal of the poset  $(\mathcal{A}_m(X), \sqsubseteq)$ . More precisely:*
  - a. *non-emptiness:*  $\perp_X \in \text{App}_m(M)$ ;
  - b. *downward closure:*  $A \in \text{App}_m(M)$  and  $A' \sqsubseteq A$  imply  $A' \in \text{App}_m(M)$ ;
  - c. *directedness:*  $\forall A_1, A_2 \in \text{App}_m(M), \exists A_3 \in \text{App}_m(M)$  such that  $A_1 \sqsubseteq A_3 \sqsupseteq A_2$ .

**Proof.** (i) By induction on a derivation of  $M \rightarrow_\beta N$ . By Remark 2, either  $M$  has a head redex or it has a head variable. If  $M = \lambda \vec{x}. (\lambda y. M') M_0 M_1 \cdots M_k$  then  $\omega_m(M) = \perp_{\text{fv}(M)}$ . By Remark 7(ii),  $M \rightarrow_\beta N$  entails  $\text{fv}(N) = \text{fv}(M)$ , whence  $\omega_m(N) \in \mathcal{A}_m(\text{fv}(M))$ . This case follows since  $\perp_{\text{fv}(M)}$  is the bottom of  $\mathcal{A}_m(\text{fv}(M))$ .

If  $M = \lambda \vec{x}. y M_1 \cdots M_k$  then  $\omega_m(M) = \lambda \vec{x}. y \omega_m(M_1) \cdots \omega_m(M_k)$ . The  $\beta$ -reduction must occur in some  $M_i \rightarrow_\beta N_i$ , whence  $\omega_m(N) = \lambda \vec{x}. y \omega_m(M_1) \cdots \omega_m(N_i) \cdots \omega_m(M_k)$ . We conclude since  $\omega_m(M_i) \sqsubseteq \omega_m(N_i)$  holds, from the induction hypothesis.

(ii) To prove that  $\text{App}_m(M)$  is an ideal, we need to check three conditions.

- a. *non-emptiness:* since  $\perp_X \sqsubseteq \omega_m(M)$  for all  $M \in \mathcal{A}_m(X)$ , we get  $\perp_X \in \text{App}_m(M)$ .
- b. *downward closure:* by definition of  $\text{App}_m(M)$ .
- c. *directedness:* let us take  $A_1, A_2 \in \text{App}_m(M)$  and prove that  $A_1 \sqcup A_2 \in \text{App}_m(M)$ .

We proceed by structural induction on, say,  $A_1$ .

Case  $A_1 = \perp_X$ . Then  $A_2 \in \mathcal{A}_m(X)$  and  $A_1 \sqcup A_2 = A_2 \in \text{App}_m(M)$ .

Case  $A_1 = \lambda x_1 \dots x_n. y A'_1 \cdots A'_k$ . If  $A_2 = \perp_X$  then proceed as before. Otherwise  $A_1 \in \text{App}_m(M)$  because there is a reduction  $M \twoheadrightarrow_\beta \lambda x_1 \dots x_n. y M'_1 \cdots M'_k =: M_1$  with  $A'_i \in \text{App}_m(M'_i)$ . Similarly,  $A_2 \in \text{App}_m(M)$  because there is a reduction  $M \twoheadrightarrow_\beta \lambda x_1 \dots x_{n'}. y M''_1 \cdots M''_{k'} =: M_2$  with  $A_2 = \lambda x_1 \dots x_{n'}. y A''_1 \cdots A''_{k'}$ , and  $A''_j \in \text{App}_m(M''_j)$ . By confluence of  $\rightarrow_\beta$ ,  $n = n'$  and  $k = k'$ , and every  $M'_i, M''_i$  have a common reduct  $M'''_i$ . By Lemma 19(i),  $A'_i, A''_i \in \text{App}_m(M'''_i)$ , so by HI  $A'_i \sqcup A''_i \in \text{App}_m(M'''_i)$ . Conclude by taking  $A_3 := \lambda x_1 \dots x_n. y (A'_1 \sqcup A''_1) \cdots (A'_k \sqcup A''_k)$ . ◀

► **Proposition 20** (recalled from Page 7). *Let  $M, N \in \Lambda_1(X)$ . If  $M \rightarrow_\beta N$ , then  $\text{App}_m(M) = \text{App}_m(N)$ .*

**Proof.** To prove  $\text{App}_m(M) = \text{App}_m(N)$ , we show the two inclusions.

( $\subseteq$ ) Take  $A \in \text{App}_m(M)$ , then there exists a reduction  $M \twoheadrightarrow_\beta M'$ , such that  $A \sqsubseteq \omega_m(M')$ .

We proceed by induction on  $A$ .

Case  $A = \perp_X$ . By Remark 17, since  $\perp_X \sqsubseteq \omega_m(N)$ , for all  $N \in \Lambda_1(X)$ .

Case  $A = \lambda \vec{x}. y A_1 \cdots A_k$ . Then  $M' = \lambda \vec{x}. y M'_1 \cdots M'_k$  with  $A_i \sqsubseteq \omega_m(M'_i)$ , for all  $1 \leq i \leq k$ . Since  $M \rightarrow_\beta N$ , by confluence,  $N$  and  $M'$  have a common reduct  $N' = \lambda \vec{x}. y N'_1 \cdots N'_k$  with  $M'_i \twoheadrightarrow_\beta N'_i$ . By Lemma 19(i) we have  $\omega_m(M'_i) \sqsubseteq \omega_m(N'_i)$  and, by transitivity,  $A_i \sqsubseteq \omega_m(N'_i)$ . We found a reduction  $N \twoheadrightarrow_\beta N'$  such that  $A \sqsubseteq \omega_m(N')$ , whence  $A \in \text{App}_m(N)$ .

( $\supseteq$ ) Straightforward, since every reduct of  $N$  is also a reduct of  $M$ .  $\blacktriangleleft$

For the proof of Theorem 21, we introduce some definitions. First, given a memory tree  $T = \text{MT}(M)$ , one defines  $\text{fv}(T)$  to be  $\text{fv}(M)$ . If  $T = \lambda x_1 \dots x_n. y^{X_1} T_1 \dots^{X_k} T_k$ , then  $\text{fv}(T) = \bigcup_{i=1}^k X_i \cup \{y\} - \{x_1, \dots, x_n\}$ . Moreover, in the following, we will consider the class of head contexts, defined inductively by the following grammar:  $H ::= [] \mid \lambda x_1 \dots x_n. y H_1 \cdots H_k$ .

Then, in order to relate memory trees with approximants, we extend the order on approximants to memory trees and define the set of finite subtrees of a memory tree:

► **Definition 48.** Let  $\mathcal{M} = \{\text{MT}(M) \mid M \in \Lambda_I\}$ . We define  $\sqsubseteq \subseteq \mathcal{M}^2$  as the least partial order on memory trees closed under the following rules:

$$\frac{\text{fv}(T) = X}{\perp_X \sqsubseteq T} \quad \frac{T_i \sqsubseteq T'_i \text{ for all } 1 \leq i \leq k}{\lambda x_1 \dots x_n. y^{X_1} T_1 \dots^{X_k} T_k \sqsubseteq \lambda x_1 \dots x_n. y^{X_1} T'_1 \dots^{X_k} T'_k}$$

► **Definition 49.** Given a memory tree  $T = \text{MT}(M)$ , one defines  $T^*$  to be  $\{T' \mid T' \sqsubseteq T \text{ and } T' \text{ is finite}\}$ .

► **Definition 50.** We define inductively the set of positions of an approximant  $A$ ,  $\text{Pos}(A)$  as a subset of  $\mathbb{N}^*$  such that:

- If  $A = \perp_X$ ,  $\text{Pos}(A) = \epsilon$ ;
- If  $A = \lambda x_1 \dots x_n. y A_1 \cdots A_k$ ,  $\text{Pos}(A) = \{i \cdot p \mid 1 \leq i \leq k \text{ \& } p \in \text{Pos}(A_i)\}$ .

Sets of positions are lifted to sets of approximants as expected and one can similarly define  $\text{Pos}(M) = \text{Pos}(\text{App}_m(M))$ .

► **Definition 51.** Given an approximant  $A$  and  $p \in \text{Pos}(A)$ , one defines  $A(p)$  by induction on the length of  $p$  as follows:

- If  $p = \epsilon$ , then if  $A = \perp_X$ ,  $A(p) = \perp_X$ , otherwise,  $A = \lambda x_1 \dots x_n. y A_1 \cdots A_k$  and  $A(p) = \lambda x_1 \dots x_n. y$ ;
- If  $p = i \cdot p'$ , then  $A = \lambda x_1 \dots x_n. y A_1 \cdots A_k$ , with  $1 \leq i \leq k$  and  $A(p) = A_i(p')$ .

► **Theorem 21** (recalled from Page 8). There is a bijection  $\mathcal{A}(\cdot)$  between the set of memory trees and the set of approximants of  $\lambda$ I-terms satisfying  $\mathcal{A}(\text{MT}(M)) = \text{App}_m(M)$ , for all  $M \in \Lambda_I$ .

**Proof of Theorem 21.** Let us write  $\mathbb{T} = \{\text{MT}(M) \mid M \in \Lambda_I\}$  and  $\mathbb{A} = \{\text{App}_m(M) \mid M \in \Lambda_I\}$  and let us build a bijection  $\mathcal{A} : \mathbb{T} \rightarrow \mathbb{A}$  such that for any  $M \in \Lambda_I$ ,  $\mathcal{A}(\text{MT}(M)) = \text{App}_m(M)$ . Let us define  $\mathcal{A}$ . Let  $M \in \Lambda_I$  and  $T$  its memory tree. Since  $T^*$  contains only finite "subtrees" of  $T$ , which are all memory trees of  $\lambda$ I-terms, one can consider a set of approximants  $\mathcal{A}(T)$  obtained as the direct image of  $T^*$  by the mapping  $\iota^-$  defined in the proof of Lemma 15.

In other words, we define  $\mathcal{A}(T) := \iota^-(T^*)$ . We now prove that  $\mathcal{A}(T) = \text{App}_m(M)$ .

Let us prove that  $\mathcal{A}(T) \subseteq \text{App}_m(M)$ . Let  $T' \in T^*$ , then there exist  $M_1 \in \Lambda_I(X_1), \dots, M_n \in \Lambda_I(X_n)$  and a head context  $H$  such that  $H[M_1, \dots, M_n] \in \Lambda_I$  with  $M \twoheadrightarrow_\beta H[M_1, \dots, M_n]$  and  $T = \text{MT}(H[\Omega \vec{X}_1, \dots, \Omega \vec{X}_n])$ . Therefore  $\iota^-(T') = H[\perp_{X_1}, \dots, \perp_{X_n}]$  and since  $\perp_{X_i} \sqsubseteq \omega_m(M_i)$ ,  $\iota^-(T') \in \text{App}_m(M)$  which proves the inclusion.

For the converse inclusion, we prove by induction on the structure of  $A$  that, for any  $M \in \Lambda_I$  and  $A \in \text{App}_m(M)$ ,  $A \in \mathcal{A}(\text{MT}(M))$ :

- If  $A = \perp_X$ , then  $X = \text{fv}(M)$  and  $A \in \iota^-(\text{MT}(M)^*)$ ;

- ─ Otherwise,  $A = \lambda x_1 \dots x_n. y A_1 \dots A_k$ . In such a case, we have  $M \rightarrow_h \lambda x_1 \dots x_n. y M_1 \dots M_k$  with  $M_i \in \Lambda_1(X_i)$  and  $A_i \in \text{App}_m(M_i)$  for  $1 \leq i \leq k$ . By induction hypothesis we have some  $T_i \in \text{MT}(M_i)^*$  such that  $A_i \in \iota^-(T_i)$  for all  $i$  ( $1 \leq i \leq k$ ). Since  $\lambda x_1 \dots x_n. y. X^1 T_1 \dots X^k T_k \in \text{MT}(M)^*$ , we indeed have that  $A \in \mathcal{A}(\text{MT}(M))$ .

Let us define  $\mathcal{T}$ , the inverse of  $\mathcal{A}$ . Let  $A, B \in \text{App}_m(M)$ . By directedness of  $\text{App}_m(M)$  (Lemma 19(ii)), there exists  $C \in \text{App}_m(M)$  such that  $A, B \sqsubseteq C$ . From this, it follows that at any position where two approximants of  $M$  are defined and different from  $\perp_X$ , they agree. We define a (potentially) infinite tree  $\mathcal{T}(\text{App}_m(M))$  from  $\text{App}_m(M)$  as a pair of partial functions  $(v, e)$ , respectively labelling the edges and vertices of the tree, defined on  $\text{Pos}(\text{App}_m(M))$  as follows:

- ─ for every position  $p \in \text{Pos}(\text{App}_m(M))$ , either for every  $A \in \text{App}_m(M)$  such that  $p \in \text{Pos}(A)$ ,  $A(p) = \perp_X$ , in which case  $v(p) = \perp_X$ , or there is some  $A \in \text{App}_m(M)$  such that  $p \in \text{Pos}(A)$ ,  $A(p) = \lambda x_1 \dots x_n. y$  in which case  $v(p) := \lambda x_1 \dots x_n. y$ .
- ─ for every non-empty position  $p \in \text{Pos}(\text{App}_m(M))$ , let  $A \in \text{App}_m(M)$  be such that  $A(p) = \perp_X$  and set  $e(p) := X$ .

A simple coinduction gives us  $\mathcal{T}(\mathcal{A}(\text{MT}(M))) = \text{MT}(M)$  for any  $M \in \Lambda_1$ . ◀

## B Proofs of Section 4

### B.1 Substitution lemma (Lemma 30)

► **Lemma 52.** *Given  $s \in (!)\Delta_1(X)$ ,  $\bar{u} \in !\Delta_1(Y)$  and  $x \notin X$ ,*

$$s \langle \bar{u}/x \rangle = \begin{cases} s & \text{if } \bar{u} = 1_Y, \\ \mathbf{0}_X & \text{otherwise.} \end{cases}$$

**Proof.** Straightforward induction on  $s$ . ◀

► **Lemma 30** (recalled from Page 10). *Given  $s \in (!)\Delta_1(X)$ ,  $\bar{u} \in !\Delta_1(Y)$ ,  $\bar{v} \in !\Delta_1(Z)$ ,  $x \in X - Z$  and  $y \in X \cup Y$ ,*

$$s \langle \bar{u}/x \rangle \langle \bar{v}/y \rangle = \sum_{\bar{v} = \bar{v}' \cdot \bar{v}''} s \langle \bar{v}'/y \rangle \langle \bar{u} \langle \bar{v}''/y \rangle /x \rangle.$$

**Proof.** By induction on  $s$ .

- ─ Case  $s = x$ . (It is the only possible case of a variable because of the hypothesis  $x \in X$ .) We have

$$\sum_{\bar{v} = \bar{v}' \cdot \bar{v}''} x \langle \bar{v}'/y \rangle \langle \bar{u} \langle \bar{v}''/y \rangle /x \rangle = x \langle 1_Z/y \rangle \langle \bar{u} \langle \bar{v}/y \rangle /x \rangle = x \langle \bar{u} \langle \bar{v}/y \rangle /x \rangle,$$

hence if  $\bar{u} = [u]$ , then both sides of the equality are equal to  $u \langle \bar{v}/y \rangle$ , otherwise they are equal to  $\mathbf{0}_{Y\{Z/y\}}$ .

- ─ Case  $s = \lambda y. s'$ . Immediate by the induction hypothesis on  $s'$ .
- ─ Case  $s = s' \bar{s}''$ . We have  $X = \text{fv}(s) = \text{fv}(s') \cup \text{fv}(\bar{s}'')$ . There are several possible cases.
  - ─ If  $x \in \text{fv}(s')$  and  $x \in \text{fv}(\bar{s}'')$ , then it is possible to apply the induction hypothesis in both subterms. By the definition of resource substitution,

$$\begin{aligned} s \langle \bar{u}/x \rangle \langle \bar{v}/y \rangle &= \sum_{\bar{u} = \bar{u}_l \cdot \bar{u}_r} \sum_{\bar{v} = \bar{v}_l \cdot \bar{v}_r} s' \langle \bar{u}_l/x \rangle \langle \bar{v}_l/y \rangle \bar{s}'' \langle \bar{u}_r/x \rangle \langle \bar{v}_r/y \rangle \\ &= \sum_{\bar{u} = \bar{u}_l \cdot \bar{u}_r} \sum_{\bar{v} = (\bar{v}'_l \cdot \bar{v}''_l) \cdot (\bar{v}'_r \cdot \bar{v}''_r)} s' \langle \bar{v}'_l/y \rangle \langle \bar{u}_l \langle \bar{v}''_l/y \rangle /x \rangle \bar{s}'' \langle \bar{v}'_r/y \rangle \langle \bar{u}_r \langle \bar{v}''_r/y \rangle /x \rangle \end{aligned}$$

by the induction hypotheses on  $s'$  and  $\bar{s}''$ ,

$$= \sum_{\bar{v}=\bar{v}' \cdot \bar{v}''} s \langle \bar{v}'/y \rangle \langle \bar{u} \langle \bar{v}''/y \rangle /x \rangle$$

by permuting the sums and re-arranging the indices (using associativity and commutativity of the multiset union).

- If  $x \in \text{fv}(s)$  and  $x \notin \text{fv}(\bar{s}'')$ , then again

$$\begin{aligned} s \langle \bar{u}/x \rangle \langle \bar{v}/y \rangle &= \sum_{\bar{u}=\bar{u}_l \cdot \bar{u}_r} \sum_{\bar{v}=\bar{v}_l \cdot \bar{v}_r} s' \langle \bar{u}_l/x \rangle \langle \bar{v}_l/y \rangle \bar{s}'' \langle \bar{u}_r/x \rangle \langle \bar{v}_r/y \rangle \\ &= \sum_{\bar{v}=\bar{v}_l \cdot \bar{v}_r} s' \langle \bar{u}/x \rangle \langle \bar{v}_l/y \rangle \bar{s}'' \langle 1_Y/x \rangle \langle \bar{v}_r/y \rangle \\ &\quad + \sum_{\substack{\bar{u}=\bar{u}_l \cdot \bar{u}_r \\ \bar{u}_r \neq 1_Y}} \sum_{\bar{v}=\bar{v}_l \cdot \bar{v}_r} s' \langle \bar{u}_l/x \rangle \langle \bar{v}_l/y \rangle \bar{s}'' \langle \bar{u}_r/x \rangle \langle \bar{v}_r/y \rangle \\ &= \sum_{\bar{v}=\bar{v}_l \cdot \bar{v}_r} s' \langle \bar{u}/x \rangle \langle \bar{v}_l/y \rangle \bar{s}'' \langle \bar{v}_r/y \rangle + \mathbf{0}_{X\{Y/x\}\{Z/y\}} \end{aligned}$$

by Lemma 52,

$$= \sum_{\bar{v}=(\bar{v}'_l \cdot \bar{v}''_l) \cdot \bar{v}_r} s' \langle \bar{v}'_l/y \rangle \langle \bar{u} \langle \bar{v}''_l/y \rangle /x \rangle \bar{s}'' \langle \bar{v}_r/y \rangle$$

by the induction hypothesis on  $s'$ ,

$$= \sum_{\bar{v}=(\bar{v}'_l \cdot \bar{v}_r) \cdot \bar{v}''_l} s \langle \bar{v}'_l \cdot \bar{v}_r/y \rangle \langle \bar{u} \langle \bar{v}''_l/y \rangle /x \rangle$$

by re-arranging the indices, and using again Lemma 52 together with the hypotheses  $x \notin Z$  and  $x \notin \text{fv}(\bar{s}'')$ . This is exactly the desired result.

- The case  $x \notin \text{fv}(s)$  and  $x \in \text{fv}(\bar{s}'')$  is symmetric.
- Case  $s = 1_X$ . If  $\bar{u} = 1_Y$  and  $\bar{v} = 1_Z$ , both sides of the equality are equal to  $1_{X\{Y/x\}\{Z/y\}}$ . Otherwise, they are equal to  $\mathbf{0}_{X\{Y/x\}\{Z/y\}}$ .
- Case  $s = s' \cdot \bar{s}''$  is similar to the case of application above. ◀

## B.2 Strong confluence of $\rightarrow_r$ (Theorem 33, item ii)

► **Lemma 53.** *If  $s \rightarrow_r s'$  then  $s \langle \bar{t}/x \rangle \rightarrow_r^? s' \langle \bar{t}/x \rangle$ .*

**Proof.** By induction on  $s \rightarrow_r s'$ . For the base case  $(\lambda y.u)\bar{v} \rightarrow_r u \langle \bar{v}/y \rangle$ ,

$$\begin{aligned} ((\lambda y.u)\bar{v}) \langle \bar{t}/x \rangle &= \sum_{\bar{t}=\bar{t}' \cdot \bar{t}''} (\lambda y.u \langle \bar{t}'/x \rangle) \bar{v} \langle \bar{t}''/y \rangle \\ &\rightarrow_r^? \sum_{\bar{t}=\bar{t}' \cdot \bar{t}''} u \langle \bar{t}'/x \rangle \langle \bar{v} \langle \bar{t}''/y \rangle /y \rangle \\ &= u \langle \bar{v}/y \rangle \langle \bar{t}/x \rangle \end{aligned} \quad \text{by Lemma 30.}$$

Notice that the reflexive closure  $\rightarrow_r^?$  of  $\rightarrow_r$  is necessary to handle the case where the two sums are empty. The other cases are immediate applications of the induction hypotheses. ◀

► **Lemma 54.** *If  $\bar{t} \rightarrow_r \bar{t}'$  then  $s \langle \bar{t}/x \rangle \rightarrow_r^? s \langle \bar{t}'/x \rangle$ .*

**Proof.** By a straightforward induction on the different cases defining  $s \langle \bar{t}/x \rangle$ . ◀

► **Lemma 55.** *For all  $s \in (!)\Delta_1(X)$  and  $s', s'' \in \mathbb{N}[(!)\Delta_1(X)]$  such that*

$$s \rightarrow_r s' \quad \text{and} \quad s \rightarrow_r s'',$$

*there exists  $t \in \mathbb{N}[(!)\Delta_1(X)]$  such that*

$$s' \rightarrow_r^? t \quad \text{and} \quad s'' \rightarrow_r^? t.$$

**Proof.** Take  $s, s'$  and  $s''$  as above. We proceed by induction on both reductions  $s \rightarrow_r s'$  and  $s \rightarrow_r s''$ . The base case is when the first reduction comes from the first rule in Definition 31, *i.e.*  $s = (\lambda x.u)\bar{v}$  and  $s' = u \langle \bar{v}/x \rangle$ :

- If the second reduction comes from the same rule, then  $s'' = s'$  and we set  $t := s'$ .
- If the second reduction comes from the rule for left application, *i.e.*  $s'' = (\lambda x.u'')\bar{v}$  with  $u \rightarrow_r u''$ , then by Lemma 53  $s' = u \langle \bar{v}/x \rangle \rightarrow_r^? u'' \langle \bar{v}/x \rangle$  and by Definition 31  $s'' \rightarrow_r^? u'' \langle \bar{v}/x \rangle$ . (The reflexive closure is needed in the latter case because  $u''$  may be empty.)
- If the second reduction comes from the rule for right application, then the proof is as in the previous case, using Lemma 54 instead of Lemma 53.

In all other cases, the results immediately follows from the induction hypotheses. ◀

► **Theorem 33** (Item ii, recalled from Page 11). *For all  $s, s', s'' \in \mathbb{N}[(!)\Delta_1(X)]$  such that*

$$s \rightarrow_r s' \quad \text{and} \quad s \rightarrow_r s'',$$

*there exists  $t \in \mathbb{N}[(!)\Delta_1(X)]$  such that*

$$s' \rightarrow_r^? t \quad \text{and} \quad s'' \rightarrow_r^? t.$$

**Proof.** We prove the result under the slightly more general hypothesis that  $s \rightarrow_r^? s'$  and  $s \rightarrow_r^? s''$ . We proceed by induction on  $\text{size}(s)$ . If  $\text{size}(s) = 0$ , *i.e.*  $s$  is empty, then both reductions must be equalities and the conclusion is trivial. Otherwise, suppose  $\text{size}(s) > 0$ . Again, if any of the two reductions turns out to be an equality, the result is immediate. Otherwise  $s \rightarrow_r s'$  and  $s \rightarrow_r s''$ , *i.e.*

- $s = s_1 + s_{\neq 1}$  and  $s' = s'_1 + s'_{\neq 1}$ , together with  $s_1 \rightarrow_r s'_1$  and  $s_{\neq 1} \rightarrow_r^? s'_{\neq 1}$ ,
- $s = s_2 + s_{\neq 2}$  and  $s'' = s''_2 + s''_{\neq 2}$ , together with  $s_2 \rightarrow_r s''_2$  and  $s_{\neq 2} \rightarrow_r^? s''_{\neq 2}$ .

There are two possible cases.

- If  $s_1 = s_2$  and  $s_{\neq 1} = s_{\neq 2}$ , then by Lemma 55 applied to  $s_1 \rightarrow_r s'_1$  and  $s_1 \rightarrow_r s''_2$ , we obtain  $t_1$  such that  $s'_1 \rightarrow_r t_1$  and  $s''_2 \rightarrow_r t_1$ . By the induction hypothesis on  $s_{\neq 1}$ , we obtain  $t_{\neq 1}$  such that  $s'_{\neq 1} \rightarrow_r t_{\neq 1}$  and  $s''_{\neq 2} \rightarrow_r t_{\neq 1}$ . Therefore we can set  $t := t_1 + t_{\neq 1}$ .
- If  $s_1 \neq s_2$ , then we can write  $s = s_1 + s_2 + s_{\neq 12}$ , with:

$$\begin{array}{llll} s_{\neq 1} = s_2 + s_{\neq 12} & s'_{\neq 1} = s'_2 + s'_{\neq 12} & s_2 \rightarrow_r^? s'_2 & s_{\neq 12} \rightarrow_r^? s'_{\neq 12} \\ s_{\neq 2} = s_1 + s_{\neq 12} & s''_{\neq 2} = s''_1 + s''_{\neq 12} & s_1 \rightarrow_r^? s''_1 & s_{\neq 12} \rightarrow_r^? s''_{\neq 12}. \end{array}$$

By Lemma 55 applied to  $s_1$  we obtain  $t_1$  such that  $s'_1 \rightarrow_r^? t_1$  and  $s''_1 \rightarrow_r^? t_1$ . By the same lemma applied to  $s_2$  we obtain  $t_2$  such that  $s'_2 \rightarrow_r^? t_2$  and  $s''_2 \rightarrow_r^? t_2$ . By the induction hypothesis on  $s_{\neq 12}$  we obtain  $t_{\neq 12}$  such that  $s'_{\neq 12} \rightarrow_r^? t_{\neq 12}$  and  $s''_{\neq 12} \rightarrow_r^? t_{\neq 12}$ . Finally, we can set  $t := t_1 + t_2 + t_{\neq 12}$ . ◀