An unconventional view on beta-reduction in name-free lambda-calculus

ninth draft version

Rob Nederpelt*and Ferruccio Guidi†

June 12, 2025

Abstract

Terms in lambda-calculus can be represented as planar trees decorated with symbols for 'abstraction' and 'application', and having variables as leafs. In this paper we concentrate on the *branches* of such trees, rather than on the trees themselves. We reformulate several well-known notions of beta-reduction in this view. In a natural manner, this reconsideration eventually leads to a new form of beta-reduction, being *expanding* – in the sense that the reduction of term t_1 to term t_2 entails that the tree of t_1 is a subtree of the tree of t_2.

1 Preliminary remarks

1.1 Introduction

It is well known that practical implementations of λ-calculus turn α-equivalence into syntactic equality by representing bound variable occurrences with depth indexes or level indexes, being positive numbers, rather than with names (de Bruijn, 1972). Therefore, the resulting systems are termed name-free as opposed to name-carrying. Generally speaking, β-reduction involves replacing the occurrences of the bound variables in the function's body with copies of the function's arguments and, in this scenario, the indexes occurring in such copies may need an update to prevent captures. Experience shows that this update, known as lift according to a well-established terminology, is a time consuming operation (Guidi, 2009, Appendix A2) that, precisely, computing machines strive to avoid (Kluge, 2005). In a family of systems originating from de Bruijn (1978b) and $\lambda\sigma$

*Eindhoven University of Technology, Dept. of Math. and Comp. Sc., P.O. Box 513, 5600 MB Eindhoven, The Netherlands

†University of Bologna, Dept. of Comp. Sc. and Engin. (DISI). Mura Anteo Zamboni 7, 40127, Bologna, Italy

(Abadi *et al.*, 1991), β-reductions insert information on updates in the copied terms. Thus a computation can delay updates at will or apply them whenever is the case.

One of the name-free system we present in this article is based just on β-reduction at a distance, an extension of β-reduction that has been introduced in Nederpelt, 1973. Such a reduction relation allows, for example, not only β-reduction $K \equiv (\lambda xy\,.\,L)MN \to (\lambda y\,.\,L[x := M])N$, but also $K \to (\lambda x\,.\,L[y := N])M$. See also Regnier, 1992, 1994.

In deviation of the usual notation, lambda terms are presented as trees composed of branches. With this representation, one obtains a transparent view on matching pairs of abstraction and application, each of which pairs may generate a beta-reduction. This transparency facilitates our discussion considerably.

1.2 About the tree structure of lambda terms

The main motivation behind this paper is sheer curiosity. It is obvious that terms in (untyped or typed) λ-calculus have a tree structure, decorated with abstraction- and application-symbols – say λ and @, respectively. In the *name-carrying* versions, the leaves are variables, in the *name-free* case these are positive natural numbers. See Figure 1, left part, for the name-free tree corresponding to the (untyped) term

$(*)$ $(\lambda((\lambda\lambda\,2\,2)\lambda 2)1)\lambda 1$,

this being the name-less version of the name-carrying term

$(\lambda x\,.\,(\lambda y\,.\,\lambda z\,.\,y\,y)(\lambda u\,.\,x)x)\lambda v\,.\,v$.

Our work has been inspired by the following question:

> *(Branch focus). Is it feasible to describe various reduction relations by concentrating not on the* **trees** *but on the* **branches** *of the trees.*

In the present paper, we try to answer this question for various well-known forms of beta-reduction. This leads us to a new form of beta-reduction having the property that it *expands* the tree under consideration, without any losses.

In this paper, we generally investigate the *name-free tree format* (firstly described in de Bruijn, 1972; see also de Bruijn, 1977).

It is obvious that without any form of order, many trees correspond to a given set of branches. (See Example 1.1, left hand side, for a linear representation of all branches in Figure 1.)

Our first goal is to ensure that *the full set of branches represents the tree it originates from*. This brings along that several issues have to be considered:

(i) *sound tree reconstruction* A tree as in Figure 1, (i), is *planar*, i.e., below every @ follow a *left* branch and a *right* branch. In *branches* containing an @, however, there is no clue whether the original path in the tree went left or right.

We have chosen to repair this by adding a 'label' S (for 'subterm') on top of the *right* branch, leaving the left branch unchanged. See Figure 1, (ii). For an

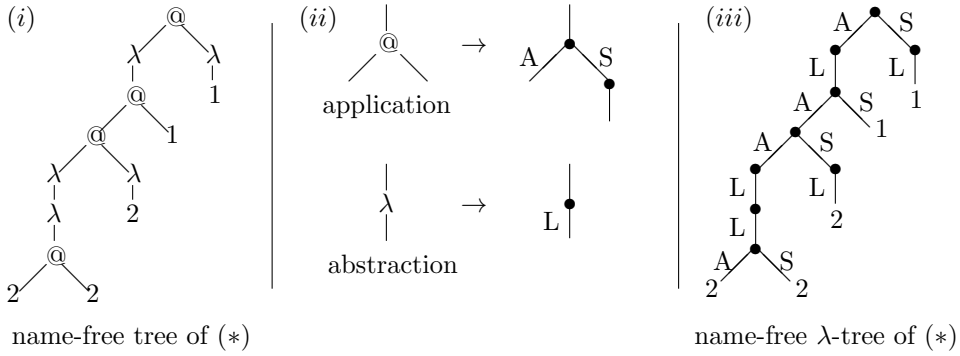

Figure 1: Name-free lambda trees; traditional and adapted

easier representation, we use A instead of @, and L instead of λ. This will be done henceforth in the paper.

(*ii*) *sound redex detection* In the tree, an @ with a λ immediately *left* below it, determines a redex. An @ with a λ immediately *right* below it, however, does *not* (Cf. Example 1.1, (3)).

The proper addition of labels S, as described right now, ensures that the difference between 'descending to the left' and 'descending to the right' has been covered by whether, no or yes, there is an S between the @ and the λ.

(*iii*) *type preparedness* The standard procedure for the detection of the λ binding a (numeric) variable n in a tree is this: one follows (upwards) the branch ending in this n and subtracts 1 for every λ met. The λ where 1 changes into 0 is the binding λ for the n. (See Figure 1, (*i*), for examples.)

Preferably, our focus on branches should be appropriate for the extension to *typed* lambda calculus (cf. Barendregt, 1992). Then there is, however, an annoying anomaly in the described search for the binding λ. To be precise: the count from n to 0, upwards along the branch, should *bypass* every λ at a bifurcation that is approached from *right* below. The reason is, that the subtree right below such a λ (representing a type) does never contain numeric variables bound by this λ. (This has been noted explicitly in de Bruijn, 1987, Section 2.) We illustrate this anomaly in Figure 2, (*i*).

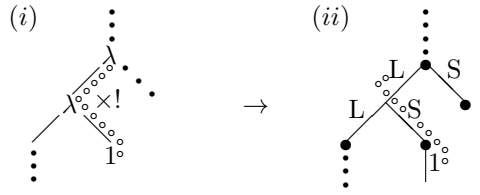

Figure 2: The binding of a variable; traditional and adapted

(*iv*) *skipping a useless* @*-sign* A similar remark as in (*iii*) holds, in both the

untyped and the typed cases, when not a λ, but an @ is approached from right below. Such an @ should actually be also skipped in the 'coding' of the branch, since this @ has no influence whatsoever.

This is awkward. Our solution to (*iv*) (and also (*iii*)!) is to lower all labels @ and λ, in the sense that they become attached to the *edge left below* the original label (so not to the vertex). Moreover, we attach the new label S to the edge *right below* the bifurcation. Apart from that, we add an edge for every numeric variable, and attach the variable to this new edge. See Figure 1, (*ii*) and also Figure 2, (*ii*).

For the skipping of an @-sign, see Example 1.1, (2), (3) and (4): compare the traditional branches and the adapted ones.

Example 1.1. The trees in Figure 2 have five branches. The lists of the labels, from root to leaf, along these branches, are the following. The branches of the tree are ordered from left to right.

	In the traditional tree:	In the adapted tree:
(1)	@ λ @ @ λ λ @ 2	A L A A L L A 2
(2)	@ λ @ @ λ λ @ 2	A L A A L L S 2
(3)	@ λ @ @ λ 2	A L A S L 2
(4)	@ λ @ 1	A L S 1
(5)	@ λ 1	S L 1

Note 1.2. *Branches (1) and (2) are* identical *in the traditional tree, but* different *in the adapted tree.*

Note 1.3. *Another advantage of the tree representation is, that there is no need for an extra marker or another technical intervention to delimit a subterm, as is often required in the process of* explicit substitution *being executed on a* linear *presentation of a λ-term. For example, let's consider an explicit substitution operator, say Σ, that we want to 'propel' one step forward through a linearly written λ-term: $\ldots \Sigma\, \mathbf{t_1}\, \mathbf{t_2} \ldots$. (We assume that subterm $\mathbf{t_1}\, \mathbf{t_2}$ is written as function $\mathbf{t_1}$* preceded *by argument $\mathbf{t_2}$; cf., de Bruijn, 1972.) Then the result of the propelling of Σ could look like $\ldots (\Sigma\, \mathbf{t_1}\, \sharp)\, \Sigma\, \mathbf{t_2} \ldots$, in which the inserted symbol $\sharp$ delimits the subterm $\mathbf{t_1}$, so that the first copy of Σ can halt in time (and becomes erased). Cf., Nederpelt, 1979, p. 7 and Nederpelt, 1980, p. 5, 6.*

1.3 Lambda trees and paths

We give the name *λ-tree* to trees of lambda terms as exemplified in the *adapted* tree of Figure 1, (*iii*). See Definition 1.4, (*i*), below. For λ-trees, we use the word *path* for a branch or a part of a branch and the word *num-label* for a numeric variable.

Definition 1.4. (*i*) A *lambda-tree* is a connected acyclic undirected graph constructed by the inductive definition below. Such a tree must be non-empty, rooted and edge-labeled.

Let $\mathbf{t}$, $\mathbf{t}_1$ and $\mathbf{t}_2$ be λ-trees and n a positive natural number. Then also the following trees are λ-trees:

Meta-variables for λ-trees: $\mathbf{t}$, $\mathbf{t}'$, $\mathbf{t}''$ ....

(*ii*) A *label* is one of A, L, S or any $n \in \mathbb{N}^+$. Meta-variables: ℓ, ...

(*iii*) A *path* in a tree $\mathbf{t}$ is a connected string of labelled edges occurring in $\mathbf{t}$. (So it may be just a *part* of a branch.) Meta-variables for paths: p, q, ...

Lemma 1.5. *Along different paths in a λ-tree one finds different strings of labels.*

Proof Let p_1 and p_2 be different paths in $\mathbf{t}$. Find the leftmost position where both paths deviate. This must be at a bifurcation. So at that position, one label is an A and the other an S. Hence, the strings of labels along p_1 and p_2 differ, as well. □

This lemma enables us to *identify* a path with its string of labels. See also Note 1.2 and Definition 1.7. We give names to special types of paths.

Definition 1.6. Let $\mathbf{t}$ be a λ-tree and p be a non-empty path in $\mathbf{t}$. Notation: $p \in \mathbf{t}$ and $p \not\equiv \varepsilon$.

p is a *root path* of $\mathbf{t}$ if p starts in the root of $\mathbf{t}$. Notation: $p \in^\wedge \mathbf{t}$.

p is a *leaf path* if it has a leaf as final label. Notation: $p \in_\vee \mathbf{t}$.

p is *complete* if it is both a root path and a leaf path. Notation: $p \in^\wedge_\vee \mathbf{t}$.

Examples of complete paths: see Example 1.1, right hand side.

Definition 1.7. Let $p_1, \ldots, p_n$ be all complete paths in a λ-tree $\mathbf{t}$. Then we identify $\mathbf{t}$ with the set $\{p_1, \ldots, p_n\}$.

Definition 1.8. (*i*) The *lenght* $|p|$ of a path p in λ-tree $\mathbf{t}$ is the number of labels (including num-labels) in p.

(*ii*) The L-*length* $\|p\|$ of a path $p \in \mathbf{t}$ is the number of labels L occurring in p.

In *namefree* λ-calculus, the *binding of variables* – as known from name-carrying lambda-calculus – is expressed by the *value* of num-label n at the end of a complete path. Such a label n represents a variable. The procedure for the establishment of the bindings between labels L and num-variables has been discussed already in the previous section, under the heading *type-preparedness*. We can now give a simple definition of binding. (Note: the path $p\,\mathrm{L}\,q\,n$ is the concatenation of path p, label L, path q and label n.)

The usual notions 'bound' and 'closed' in lambda-calculus are covered by the following definition.

Definition 1.9. (*i*) Let $\mathbf{t}$ be a λ-tree and $p\,\mathrm{L}\,q\,n \in^{\wedge}_{\vee} \mathbf{t}$ such that $\|q\| = n+1$. Then this n is *bound* by the mentioned L. Moreover, the path $\mathrm{L}\,q\,n$ is called the L-*block* of (this occurrence of) n.

(*ii*) The λ-tree $\mathbf{t}$ is *closed* if all num-variables in $\mathbf{t}$ are bound by some $\mathrm{L} \in \mathbf{t}$.

Note that the binding L of a num-variable n always occurs in the (unique) complete path ending in n.

Lemma 1.10. (*i*) *The* L-*block of a certain* $n \in \mathbf{t}$*, if it exists, is unique.*

(*ii*) *In a closed term, each num-variable* n *corresponds to exactly one* L-*block; but even when the term is closed, not every* L-*block binds some num-variable.*

To every λ-tree belongs a well-defined set $\mathcal{S}$ of paths. A natural question is: when does a given set $\mathcal{S}$ of paths define a λ-tree that can be constructed according to Definition 1.4?

Since this definition is inductive, it is no surprise that a direct procedure for deciding this question is inductive, as well. We now give a verbal representation of such a procedure.

Procedure 1.11. Firstly, we require that an arbitrary path in $\mathcal{S}$ consist of members of {L, A, S} only, with as exception the final label of such a path, which must be a positive natural number. We call such a path a *proper* path.

So we may assume that $\mathcal{S}$ consists of proper paths. We further assume that $\mathcal{S}$ is finite and that all paths in $\mathcal{S}$ are finite, as well. Moreover, in order to simplify the description of the procedure that we give below, we assume that the paths are lexicographically ordered, on the basic order – say – $\mathrm{L} < \mathrm{A} < \mathrm{S}$. We number the paths accordingly: $p_1, p_2, \ldots, p_n$.

Here comes the procedure for such a set $\mathcal{S}$ of proper paths:

case 1: Let $p_1 \equiv \mathrm{L}\ q_1$ for some q_1. Then

Requirement 1 For all $1 \leq i \leq n:\ p_i \equiv \mathrm{L}\ q_i$ for some q_i.

Skip the front-Ls in all these paths and collect them: $\mathcal{S}' \equiv \{q_1, q_2, \ldots q_n\}$. Apply the procedure to $\mathcal{S}'$.

case 2: Let $p_1 \equiv \mathrm{A}\ q_1$ for some q_1. Then

Requirement 2 There must be some p_i such that $p_i \equiv \mathrm{S}\,q_i$ and there is no p_j with $p_j \equiv n\,\ell$ where $\ell \equiv \mathrm{L}$ or some n.

By the lexicographical ordering, there now must be an $1 \leq m \leq n$ such that all p_k with $k \leq m$ begin with A, and all p_k with $k > m$ begin with S.

Divide the set $\mathcal{S}$ into two parts:

$\mathcal{S}_1 := \{p_k \in \mathcal{S} \mid p_k \text{ begins with A}\}$, and $\mathcal{S}_2 := \{p_k \in \mathcal{S} \mid p_k \text{ begins with S}\}$.

Skip the front-As in all paths of $\mathcal{S}_1$ and the front-Ss in $\mathcal{S}_2$ and collect them. We obtain $\mathcal{S}'_1$ and $\mathcal{S}'_2$. Apply the procedure to $\mathcal{S}'_1$ and $\mathcal{S}'_2$.

case 3: Let $p_1 \equiv n$ for some positive n. Then

Requirement 3 $\mathcal{S} \equiv \{p_1\}$.

If one of the requirements is not met, we abort the procedure and give the answer 'no'. It is not hard to show that this procedure ends without abortion (and the answer is 'yes') if and only if the original $\mathcal{S}$ is the set of all paths belonging to one specific λ-tree. □

2 β-reduction

2.1 A short history of updating in name-free beta-reduction

In name-carrying systems of λ-calculus a binder of a term M, say λ_x, and the variable occurrences that refer to it carry the same name, say x. On the contrary, name-free systems use unnamed binders, say λ, and replace a bound variable occurrence x with an index that is a non-negative integer denoting the position of the corresponding λ_x along the path connecting x to the root of M in the representation of M as an abstract syntax tree. As we pointed out in the introduction, the β-reduction step of the latter systems requires updating the indexes occurring in a copied argument, say N, to maintain the relationship between the bound variable instances in M and the respective binders. Depending on the particular system, if *immediate updating* is in effect, the update occurs by applying a so-called *update function* to the indexes in N. On the contrary, if *delayed updating* is in effect, the update function is just stored in the syntax of the copied N.

The first name-free systems with immediate updating appear in de Bruijn (1972) with the basic update functions $\tau_{d,h}$ of type $\mathbb{N}^+ \to \mathbb{N}^+$, where $d \in \mathbb{N}$ and $h \in \mathbb{N}$.

$$\tau_{d,h} \equiv i \mapsto \begin{cases} i & \text{if } i \leq d \\ i+h & \text{if } i > d \end{cases}$$

The systems accompanying de Bruijn, 1978b – for instance those in de Bruijn, 1977, 1978a – are the first to allow delayed updating by featuring the term node $\phi(f)$ where f is an arbitrary function of type $\mathbb{N}^+ \to \mathbb{N}^+$. The original purpose of $\phi(f)$ is to present substitution as a single operation defined by recursion on the structure of terms.

Other systems of the same family, like Nederpelt, 1979, 1980; Kamareddine and Nederpelt, 1993, feature the term node $\mu(d,h)$ or $\phi^{(d,h)}$ that holds the function $\tau_{d,h}$. Moreover, the systems originating from Abadi *et al.*, 1991 – for instance those in Curien, Hardin and Lévy, 1996 – feature the explicit substitution constructors *id* and $\uparrow$ that essentially hold the functions $\tau_{0,0}$ (the identity) and $\tau_{0,1}$ (the successor) respectively.

2.2 The usual β-reduction in the path-approach

We continue with a number of useful definitions for name-free λ-calculus with emphasis on paths.

Definition 2.1. Let $\mathbf{t}$ be a λ-tree and p a fixed root path in $\mathbf{t}$.

The set of all paths $q \in_\vee \mathbf{t}$ such that $p\,q$ is a complete path in $\mathbf{t}$, is denoted by *tree*(p). We call the set $\{p\,q \mid q \in_\vee \mathit{tree}(p)\}$ the *grafted tree* of p in $\mathbf{t}$.

We note that the grafted tree of a root path p in a *closed* $\mathbf{t}$ is 'closed' itself, in the sense that all free variables in *tree*(p) are bound in p.

We shall now describe the usual β-reduction in terms of paths and grafted trees. We start with the well-known notion 'redex' (reducable expression).

Definition 2.2. Let $\mathbf{t}$ be a λ-tree. Let $p\,\mathrm{A}\,\mathrm{L} \in^\wedge \mathbf{t}$. Then the adjacent pair A L at the end of this path, identifies a *redex*. This redex consists of two elements: (1) the *'function'* L *tree*$(p\,\mathrm{A}\,\mathrm{L})$ and (2) the *'argument'* *tree*$(p\,\mathrm{S})$.

See Figure 3, (i).

We now consider the usual relation called β-reduction and expressed with the symbol $\rightarrow_\beta$. This β-reduction formalizes the action: 'apply a function to an argument'. In name-free lambda-calculus, which is our subject here, 'function application' by means of β-reduction has important consequences for the numbers acting as num-variables. Some of these numbers should be 'updated' after the β-reduction.

Definition 2.3. Let $\mathbf{t}_1$ be a λ-tree. Assume that $p\,\mathrm{A}\,\mathrm{L} \in^\wedge \mathbf{t}_1$. (For reference' sake, we call the A and the L in this path *pivotal*.) Consider the corresponding grafted tree $p\,\mathrm{A}\,\mathrm{L}\,$*tree*$(p\,\mathrm{A}\,\mathrm{L})$. Then $\mathbf{t}_1 \rightarrow_\beta \mathbf{t}_2$, where $\mathbf{t}_2$ is the tree obtained from $\mathbf{t}_1$ by

(i) substituting and *updating* (see below) *tree*$(p\,\mathrm{S})$ for every num-variable in *tree*$(p\,\mathrm{A}\,\mathrm{L})$ that is bound by the pivotal L,

(ii) erasing the pivotal A-L-pair, and

(ii) erasing all complete paths in the grafted tree $p\,\mathrm{S}\,$*tree*$(p\,\mathrm{S})$.

Definition 2.4. *Updating* num-variables due to β-reduction is the process illustrated in Figure 3.

Theorem 2.5. *Updating preserves the bond between num-variables and their binding* L*-labels.*

Proof We illustrate what happens under β-reduction in Figure 3. We display in part (i) of that figure the essential parts of the redex. In part (ii) of the same picture we show how the updating works.

We justify the preservation of the bindings in the update process as follows. Hereby we denote, for easy reference, an L binding n as L_n and an L binding l as L_l. We also use the symbol $\rightarrow_\beta$ in an unorthodox manner. We write m^{upd} for an updated m.

It suffices to inspect two representative paths:

(i) in $p\,\mathrm{A}\,\mathrm{L}\,$*tree*$(p\,\mathrm{A}\,\mathrm{L})$ we choose $p\,\mathrm{A}\,\mathrm{L}\,q\,n$,

(ii) in $p\,\mathrm{S}\,$*tree*$(p\,\mathrm{S})$ this is $p\,\mathrm{S}\,r\,l$.

We discern three cases for n:

(1) $n < \|q\|{+}1$. Then $p\,\mathrm{A}\,\mathrm{L}\,q\,n = p\,\mathrm{A}\,\mathrm{L}\,q_1\,\mathrm{L}_n\,q_2\,n \rightarrow_\beta p\,q_1\,\mathrm{L}_n\,q_2\,n^{\mathrm{upd}}$, hence $n^{\mathrm{upd}} = n$.

(2) $n > \|q\|{+}1$. Then $p\,\mathrm{A}\,\mathrm{L}\,q\,n = p_1\,\mathrm{L}_n\,p_2\,\mathrm{A}\,\mathrm{L}\,q\,n \rightarrow_\beta p_1\,\mathrm{L}_n\,p_2\,q\,n^{\mathrm{upd}}$, hence $n^{\mathrm{upd}} = n - 1$ (since the pivotal L has been erased).

(3) $n = \|q\|{+}1$. Then the pivotal L binds the n. Now we have to distinguish two cases for l:

(3a) $l \leq r$. Then $p\,\mathrm{S}\,r\,l = p\,\mathrm{S}\,r_1\,\mathrm{L}_l\,r_2\,l$, hence $p\,\mathrm{A}\,\mathrm{L}\,q\,n \rightarrow_\beta p\,q\,r_1\,\mathrm{L}_l\,r_2\,l^{\mathrm{upd}}$, so $l^{\mathrm{upd}} = l$.

(3b) $l > r$. Then $p\,\mathrm{S}\,r\,l = p_1\,\mathrm{L}_l\,p_2\,\mathrm{S}\,r\,l$, hence $p\,\mathrm{A}\,\mathrm{L}\,q\,n \rightarrow_\beta p_1\,\mathrm{L}_l\,p_2\,q\,r\,l^{\mathrm{upd}}$, so $l^{\mathrm{upd}} = l + \|q\|$, since q now pops up between L_l and r. $\square$

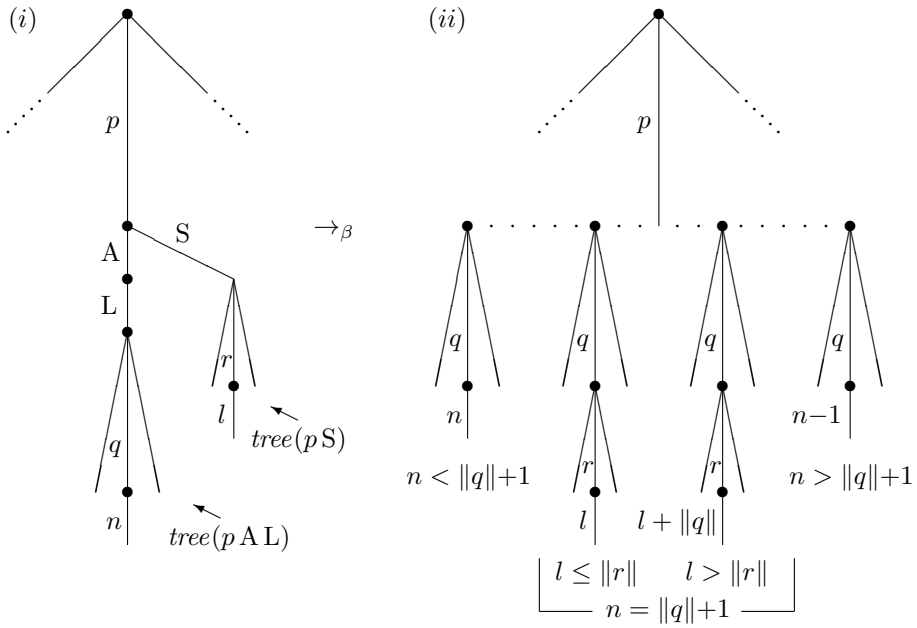

Figure 3: A picture of name-free β-reduction with updating

2.3 Comparing β-reduction in name-carrying and name-free λ-calculus

The set of terms λ-calculus that we exposed until now, with focus on paths, we denote as $\mathcal{T}^{fre}$. In the present section we compare it with the name-carrying λ-calculus with the same focus on paths, that we call $\mathcal{T}^{car}$. We do not explain how the terms in $\mathcal{T}^{car}$ look like. We assume that the reader can easily devise that. The most important differences with $\mathcal{T}^{fre}$ are:

(1) $\mathcal{T}^{car}$ has actual variables (such as x, $y\ldots$) instead of num-variables.

(2) Every L-label in $\mathcal{T}^{car}$ has a variable as subscript, e.g., L_x for some x.

And thus an L-block in $\mathcal{T}^{car}$ appears as $\mathrm{L}_x\, p\, x$ instead of $\mathrm{L}\, p\, n$. And so on. In particular, we do not repeat how binding works in $\mathcal{T}^{car}$.

Note that we assume that all λ-trees in $\mathcal{T}^{car}$ are closed and that in a λ-tree $\mathbf{t}$, all bound variables are different.

In the remainder of this Section we present a number of simple results about related facts, concerning $\mathcal{T}^{car}$ and $\mathcal{T}^{fre}$.

Most importantly, there is a well-known *isomorphism* between β-reductions in $\mathcal{T}^{car}$ and $\mathcal{T}^{fre}$. See Lemma 2.10 below. Firstly, we define mappings between $\mathcal{T}^{car}$ and $\mathcal{T}^{fre}$ and vice versa.

Procedure 2.6. Let $\mathbf{s} \in \mathcal{T}^{car}$. Then we obtain $[\mathbf{s}] \in \mathcal{T}^{fre}$ by the following method.

(i) Let x be a variable in $\mathbf{s}$, bound in $\mathcal{T}^{car}$ via the L-block $\mathrm{L}_x\, p\, x$. Replace this x by $\|p\|+1$. Do this for all num-labels.

(ii) Erase all subscripts, such as x, below labels L_x in $\mathbf{s}$.

Procedure 2.7. Let $\mathbf{t} \in \mathcal{T}^{fre}$. Then we obtain $\langle \mathbf{t} \rangle \in \mathcal{T}^{car}$ as follows.

For each label $\mathrm{L} \in \mathbf{t}$, find all num-variables $n_1, \ldots, n_k$ bound by this L. (If the mentioned L occurs as final label in the path $p\,\mathrm{L} \in^{\wedge} \mathbf{t}$, then the bound num-variables occur in $tree(p\,\mathrm{L})$.)

Now replace L by L_x, using a *new* variable x (i.e., a variable which has not yet been used in this procedure), and (if the number of bound n_i's is not zero) replace each of these n_i by x.

We extend α-equivalence to $\mathcal{T}^{fre}$. We assume that the reader understands what 'correspond' means in the following Definition and Lemmas.

Definition 2.8. (i) Let $\mathbf{t}$ be a λ-tree in either $\mathcal{T}^{car}$ or $\mathcal{T}^{fre}$. We obtain the *variable-free tree* of $\mathbf{t}$ by stripping all variables and num-labels, including the subscripts of labels $\mathrm{L}_x \in \mathcal{T}^{car}$. All other labels and all edges, including the edges that had a num-label as label, stay as they are.

(ii) Let $\mathbf{t} \in \mathcal{T}^{car}$ and $\mathbf{t}'$ a λ-tree in either $\mathcal{T}^{car}$ or $\mathcal{T}^{fre}$. Then $\mathbf{t}$ is *α-equivalent* to $\mathbf{t}'$ (in symbols: $\mathbf{t} \equiv_\alpha \mathbf{t}'$) if the variable-free trees of $\mathbf{t}$ and $\mathbf{t}'$ are identical and the bindings in $\mathbf{t}$ correspond one-to-one to the bindings in $\mathbf{t}'$.

Lemma 2.9. (i) *Mappings* $[-]$ *and* $\langle - \rangle$ *are each others inverses.*

(ii) Let $\mathbf{s} \in \mathcal{T}^{car}$ *and* $\mathbf{t} \in \mathcal{T}^{fre}$. *Then* $\mathbf{s} \equiv_\alpha [\mathbf{s}]$ *and* $\mathbf{t} \equiv_\alpha \langle \mathbf{t} \rangle$.

Now we show that the mapping $[\ ]$ from $\mathcal{T}^{car}$ to $\mathcal{T}^{fre}$ 'preserves' the binding relation between variables and L-labels.

Lemma 2.10. *Let* $\mathbf{s}_1 \in \mathcal{T}^{car}$ *and* $\mathbf{s}_1 \to_\beta \mathbf{s}_2$ *via the* β*-reduction with pivot* L_x. *Then there is a corresponding* β*-reduction* $[\mathbf{s}_1] \to_\beta [\mathbf{s}_2]$ *in* $\mathcal{T}^{fre}$, *with corresponding pivot* L.

Proof (1) In the reduction $\mathbf{s}_1 \to_\beta \mathbf{s}_2$ *the bindings are preserved (common knowledge).*

(2) $\mathbf{s}_1 \equiv_\alpha [\mathbf{s}_1]$ *(Lemma 2.9, (ii)).*

(3) In the reduction $[\mathbf{s}_1] \to_\beta [\mathbf{s}_2]$ *the bindings are preserved as well (Theorem 2.5). Moreover, the variable-free trees of* $\mathbf{s}_2$ *and* $[\mathbf{s}_2]$ *are identical (follows from the construction procedures for* $[\mathbf{s}_2]$*). So* $\mathbf{s}_2 \equiv_\alpha [\mathbf{s}_2]$.

Accordingly, there is a corresponding lemma for the inverted situation. We shall not go into it.

3 Alternative β-reductions

3.1 Balanced β-reduction in name-free λ-calculus

There is a variant of β-reduction that is interesting if it is advantageous to keep all the information that is present in the original λ-calculus term. Then an argument $tree(p\,\mathrm{S})$ should remain in the β-reduced term, just as the pivotal A-L-pair

(see Definition 2.1 and 2.2). We call this reduction relation *balanced β-reduction*. In the literature, it originally appeared under the name β_1 (Nederpelt, 1973). For details, see the more recent literature about the Linear Substitution Calculus (cf. Accattoli and Kesner, 2010 and Accattoli and Kesner, 2012), in which it is called *distant beta*, symbol $\rightarrow_{dB}$. See also Barenbaum and Bonelli, 2017 and Kamareddine and Bloo, 2005.

We start with the definition of a *balanced path* in a λ-tree.

Definition 3.1. A path p in a λ-tree $\mathbf{t}$ is called *balanced*, denoted $bal(p)$, if it is constructed by means of the following inductive rules:
(i) $bal(\varepsilon)$, i.e., the empty string is balanced;
(ii) if $bal(p)$, then $bal(\mathrm{A}p\,\mathrm{L})$;
(iii) if $bal(p)$ and $bal(q)$, then $bal(p\,q)$.
In case (ii), we say that the mentioned A *matches* the mentioned L.

Examples of balanced paths: ε, A L, A A L L, A L A L, A A L A A L L L.

Note the close correspondence between balanced paths and (consecutive) nested pairs of parentheses. Note that only A- and L-labels occur in balanced paths, so there is no other label involved, such as S.

Now maintenance of the pivotal A-L-pair in λ-tree $\mathbf{t}$, as mentioned above, has a serious consequence: it possibly prevents other instances of β-reduction, that arise in a 'normal' β-reduction. If, for example, the underlined pair $\underline{\mathrm{A\,L}}$ in the path $p\,\mathrm{A}\,\underline{\mathrm{A\,L}}\,\mathrm{L} \in^{\wedge} \mathbf{t}$ is the pivotal pair, then the maintenance of this pair prevents that the other A and the other L appear as a new pivotal pair after the one-step β-reduction induced by $\underline{\mathrm{A\,L}}$. With 'normal' β-reduction, this does not happen, since $\underline{\mathrm{A\,L}}$ then disappears.

This situation can be avoided by using *balanced* β-reduction.

The following definition is an introduction to the notion 'balanced reduction'.

Definition 3.2. Let $\mathbf{t}$ be a λ-tree, let $b \in \mathbf{t}$ be a balanced path and assume that $p\,\mathrm{A}\,b\,\mathrm{L} \in^{\wedge} \mathbf{t}$. This root path is called *active* if there is at least one path $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n \in^{\wedge}_{\vee} \mathbf{t}$ such that n is bound by the L behind b. If there is no such path, the root path is *inactive*.

Now we give the 'balanced' variant of β-reduction, with symbol $\rightarrow_b$. We recall that '$\mathrm{L}\,q\,n$ is an L-block' is equivalent to 'n is bound by the initial L'.

Definition 3.3. Let $\mathbf{t}$ be a λ-tree, let $b \in \mathbf{t}$ be a balanced path and assume that $p\,\mathrm{A}\,b\,\mathrm{L} \in^{\wedge} \mathbf{t}$ is an active root path. Let $\mathbf{t}'$ be $\mathbf{t}$ in which all paths $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n$ with $\mathrm{L}\,q\,n$ being an L-block, have been replaced by $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,tree(p)$.
Then $\mathbf{t} \rightarrow_b \mathbf{t}'$.

The condition that $p\,\mathrm{A}\,b\,\mathrm{L}$ is *active* in this definition avoids an infinite reduction path generated by the mentioned root path.

Definition 3.4. (i) The displayed L in Definition 3.3 is called the *pivotal* L.
(ii) Let $\mathbf{t}$ be a λ-tree with $r = p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n \in \mathbf{t}$, where b is a balanced path and such that the $\mathrm{L} \in r$ binds the n. Then $\mathrm{L}\,q\,n$ is called an L-block (Definition 1.9). We call $\mathrm{A}\,b\,\mathrm{L}\,q\,n$ an A-*block*.

Consider two λ-trees $\mathbf{t}$ and $\mathbf{t}'$ such that $\mathbf{t} \to_b \mathbf{t}'$ as described in Definition 3.3, so each n bound by the pivotal L has been replaced by $\mathit{tree}(p\,S)$ in $\mathit{tree}(p\,\mathrm{A}\,b\,\mathrm{L})$. Now we have that $\mathbf{t}$ is (almost) a subtree of $\mathbf{t}'$, provided that we omit all num-variables n in $\mathbf{t}$ bound by the pivotal L and omit the corresponding edges, as well. So, balanced β-reduction has the property that it *extends* the original underlying tree $\mathbf{t}$, but for a number of num-variables that disappear.

3.2 Focused β-reduction in name-free λ-calculus

Focused β-reduction is a special case of balanced β-reduction. This reduction concentrates on *precisely one* num-variable n at a specific position in a certain λ-tree $\mathbf{t}$, this n being bound by a *pivotal* L. Since L is pivotal, there must be an A 'coupled' to L, so n is the final label of a particular path $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n \in^{\wedge}_{\vee} \mathbf{t}$, where b is balanced. Focused β-reduction replaces this n by the argument connected to the pivot. See the following definition, in which the symbol $\to_f$ is introduced for 'focused' β-reduction.

Definition 3.5. Let $\mathbf{t}$ be a λ-tree, let b be a balanced path and let $r = p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n \in^{\wedge}_{\vee} \mathbf{t}$ be a fixed complete path in $\mathbf{t}$. Let $\mathbf{t}'$ be identical to $\mathbf{t}$, except that r has been replaced by $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,\mathit{tree}(p\,\mathrm{S})$.

Then $\mathbf{t} \to_f \mathbf{t}'$.

It will be clear that we want a kind of β-reduction here that preserves the A-L-pair, because there may be other num-variables bound to this L, and maybe one desires later to replace one or more of these by $\mathit{tree}(p\,\mathrm{A})$, in subsequent $\to_f$-reductions.

The motivation for introducing focused β-reduction comes from the process known as *definition unfolding* in *name-carrying* λ-calculus. Then a defined notion occurring in M, say x, is replaced by the definiens, say P. Such an action generally occurs for only one instance of the definiendum x. So instead of replacing *all* occurrences of x in M, one aims at *precisely one* occurrence.

The 'name' of the definiendum is important here, since it is hard to work with a 'name-less' definiendum. Nevertheless, we address this variant of β-reduction here, since the name-less variant is interesting as such.

The possibility of having *balanced* β-reduction is necessary to be able to deal with other A-L-pairs, which otherwise would be inaccessible. See the following example.

Example 3.6. We have, in λ-calculus with normal untyped β-reduction:

$(\lambda x\,.\,((\lambda y\,.\,M)Q))P \to_\beta (\lambda x\,.\,(M[y := Q]))P \to_\beta M[y := Q][x := P]$.

In *focused* β-reduction, this becomes:

$(\lambda x\,.\,((\lambda y\,.\,M)Q))P \to_f (\lambda x\,.\,(\lambda y\,.\,M[y_0 := Q])Q)P \to_f$

$(\lambda x\,.\,((\lambda y\,.\,M[y_0 := Q])Q)[x_0 := P])P$.

Here y_0 and x_0 are selected instances of the free y's and x's in M, respectively.

The second of the two one-step focused reductions would not be possible without the possibility to have a balanced λ-term $(\lambda y \ldots)Q$ between the λx and the P.

The following lemma is obvious.

Lemma 3.7. *Let* $\mathbf{t} \in \mathcal{T}^{fre}$ *and* $\mathbf{t} \to_b \mathbf{t}'$. *Then* $\mathbf{t} \twoheadrightarrow_f \mathbf{t}'$.

3.3 Erasing reduction

After having applied balanced or focused β-reduction, one also desires a reduction that gets rid of the 'remains', i.e., the A and the L in grafted trees $p\,\mathrm{A}\,b\,\mathrm{L}\,\mathit{tree}(p\,\mathrm{A}\,b\,\mathrm{L})$ when no $n \in \mathit{tree}(p\,\mathrm{A}\,b\,\mathrm{L})$ is bound to the displayed L. Moreover, the 'argument' $\mathit{tree}(p\,\mathrm{S})$ must be removed together with the mentioned S. We call the corresponding reduction *erasing* reduction and use symbol $\to_e$ for it. (This reduction is also referred to as 'garbage collection' in the literature; see, e.g., Rose, 1993.)

The definition of $\to_e$ is not easy, due to the fact that the erasure applies to different parts of the original tree $\mathbf{t}$.

Definition 3.8. Let $\mathbf{t}$ be a λ-tree and assume that a certain $p\,\mathrm{A}\,b\,\mathrm{L} \in^{\wedge} \mathbf{t}$, where b is balanced. Assume moreover that no num-variable in $\mathit{tree}(p\,\mathrm{A}\,b\,\mathrm{L})$ is bound by the mentioned L.

Then $\mathbf{t} \to_e \mathbf{t}'$, where $\mathbf{t}'$ is $\mathbf{t}$ in which $\mathrm{S}\,\mathit{tree}(p\,\mathrm{S})$ has been removed and in which $\mathit{tree}(p)$ has been replaced by $\mathit{tree}(p\,\mathrm{A})$ in which $\mathit{tree}(p\,\mathrm{A}\,b)$, in its turn, has been replaced by $\{q\,n \in^{\wedge}_{\vee} \mathit{tree}(p\,\mathrm{A}\,b\,\mathrm{L}) \mid n \text{ replaced by } n-1 \text{ if } n > \|q\|\}$.

The necessity to replace n by $n-1$ is, of course, caused by the erasure of the mentioned L.

Repeated application of $\to_e$ will result in a λ-tree with no garbage.

3.4 Theorems

Lemma 3.9. *Reduction* $\to_e$ *is strongly normalizing, with unique normal form.*

We denote the transitive closure of a reduction $\to_i$ by $\twoheadrightarrow_i$. An arbitrary sequence of reductions $\to_i$ and $\to_j$ is denoted $\twoheadrightarrow_{i,j}$.

Theorem 3.10. *Let* $\mathbf{t}$ *and* $\mathbf{t}'$ *be* λ*-trees.*

(*i*) $\mathbf{t} \twoheadrightarrow_\beta \mathbf{t}' \to \mathbf{t} \twoheadrightarrow_{b,e} \mathbf{t}'$.

(*ii*) $\mathbf{t} \to_b \mathbf{t}' \to \mathbf{t} \twoheadrightarrow_f \mathbf{t}'$

(*iii*) (Postponement of $\to_e$ after $\to_b$) *If* $\mathbf{t} \twoheadrightarrow_{b,e} \mathbf{t}'$, *then there is* $\mathbf{t}''$ *such that* $\mathbf{t} \twoheadrightarrow_b \mathbf{t}'' \twoheadrightarrow_e \mathbf{t}'$.

(*iv*) (Postponement of $\to_e$ after $\to_f$) *If* $\mathbf{t} \twoheadrightarrow_{f,e} \mathbf{t}'$, *then there is* $\mathbf{t}''$ *such that* $\mathbf{t} \twoheadrightarrow_f \mathbf{t}'' \twoheadrightarrow_e \mathbf{t}'$.

Proof (*iii*) Nederpelt, 1973, p. 48, Theorem 6.19.

(*iv*) Similarly. □

Theorem 3.11. $\to_b$, $\to_f$ *and* $\to_e$ *are confluent.*

Proof For $\to_b$ and $\to_e$, see Nederpelt, 1973, Theorems 6.38 and 6.42. For $\to_f$, see Accattoli and Kesner, 2012.

4 A new, loss-free β-reduction

4.1 Expanding β-reduction

The system we are going to introduce takes a simpler approach than the one mentioned in Section 2.1. Its syntax has a term node p we call *inner numeric label* where $p \in \mathbb{N}^+$. An active p holds the function $\tau_{0,p}$ while a passive, i.e. present but ignored, p holds the function $\tau_{0,0}$. In some sense, we want to show that supporting the functions $\tau_{0,h}$ suffices to implement delayed updating in basic name-free λ-calculus.

At the end of Section 3.1 we mentioned that, when $\mathbf{t} \to_b \mathbf{t}'$, the λ-tree $\mathbf{t}$ is almost a subtree of $\mathbf{t}'$. The word 'almost' concerns the fact that num-variables bound by the pivotal $\mathrm{L} \in \mathbf{t}$ are removed in the balanced reduction, so they do not reappear in $\mathbf{t}'$.

In the present section we investigate what happens if we *leave the num-variables bound by* L *where they are*. In that case $\mathbf{t}$ becomes a proper subtree of $\mathbf{t}'$. We might say that the resulting reduction has the property that *no information from* $\mathbf{t}$ *has been lost* in the reduction from $\mathbf{t}$ to $\mathbf{t}'$.

In order to make this work, we have to extend our notion of 'path': now num-variables may appear everywhere *inside* a path, so not only at the end.

Definition 4.1. An *extended* path is a finite string of labels L, A, S and arbitrary num-labels, ending in a num-label.

Example: L L 2 A A 1 A L L 2.

Now num-labels come in two sorts: inside a path or at the end. We also obtain a new kind of λ-trees.

Definition 4.2. (*i*) Num-variables not being end-labels, we call *inner num-labels*. Num-variables that *are* end-labels (leafs), we refer to as *outer* num-labels.

(*ii*) A λ-tree in which inner variables are allowed, we call an *extended* λ-tree.

Consequently, the definition of a *balanced path* (Definition 3.1) must be adapted as well, such that it allows inner num-labels *inside* the string of A's and L's.

In the remainder of this section, we assume that these new definitions of path, balanced path and λ-tree are valid. Moreover, we shall omit the word extended *for the new paths and λ-trees.*

We recall from Section 2.3 that symbol $\mathcal{T}^{fre}$ concerns the set of name-free, closed trees (without inner variables). The set of name-free, closed (extended) λ-trees where also inner variables are permitted, we denote by $\mathcal{T}^{exp}$.

Obviously, $\mathcal{T}^{fre} \subseteq \mathcal{T}^{exp}$.

The β-like reduction being a consequence of this extension with inner variables, we call *expanding* β-reduction. Again, this reduction has two obvious

flavours: *balanced* or *focused*. We concentrate from now on at *focused* reduction, since this reduction can be used to simulate balanced reduction (cf. Theorem 3.10, (ii)).

We use the symbol '$\to_{ef}$' for expanding, focused β-reduction. Its definition is as follows.

Definition 4.3. Let $\mathbf{t} \in \mathcal{T}^{exp}$, let $b \in \mathbf{t}$ be a balanced path, assume that $r = p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n \in^{\wedge}_{\vee} \mathbf{t}$ is a fixed, complete path in $\mathbf{t}$, where n is bound by the displayed L. Let $\mathbf{t}'$ be identical to $\mathbf{t}$, except that r has been replaced by $p\,\mathrm{A}\,b\,\mathrm{L}\,q\,n\,tree(p\,\mathrm{S})$.

Then $\mathbf{t} \to_{ef} \mathbf{t}'$.

(Note that the balanced path b may contain inner num-variables.)

The effect of $\to_{ef}$-reduction is that the end-label n *and* the edge labelled n stay where they are, and $tree(p\,\mathrm{S})$ is simply attached to this n. We shall see that the remaining presence of the label n enables updating at a later stage.

For a pictorial representation, see Figure 4. Note: if $tree(p\,\mathrm{S})$ consists of a single edge only, labelled with a num-variable, then this edge is just attached to the edge labelled n.

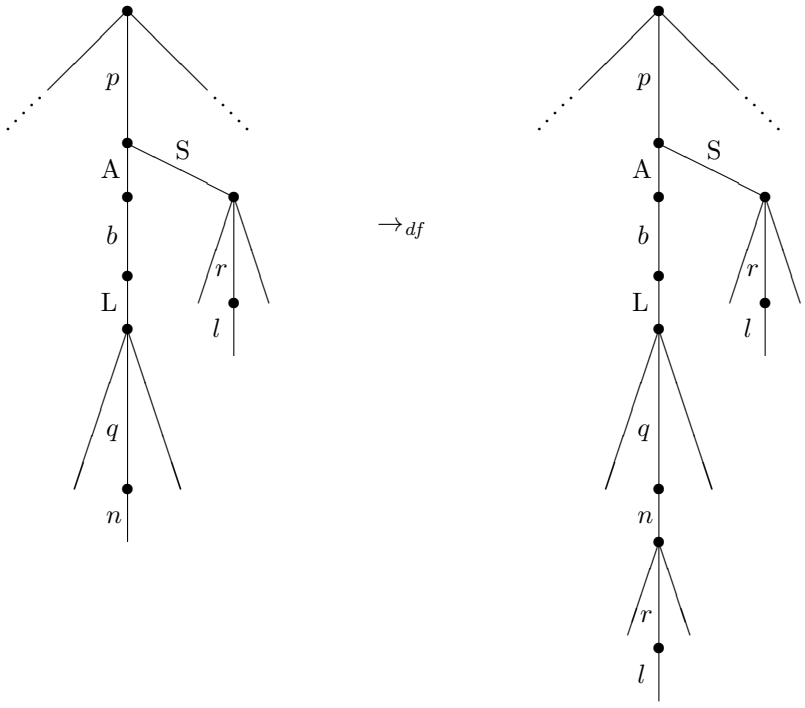

Figure 4: A picture of name-free, expanding β-reduction

We define what *inclusion* of (extended) λ-trees means.

Definition 4.4. Let $\mathbf{t}$ and $\mathbf{t}'$ be (extended) λ-trees. Then $\mathbf{t} \subseteq \mathbf{t}'$ iff $p \in^{\wedge}_{\vee} \mathbf{t} \to p \in^{\wedge} \mathbf{t}'$. Moreover, $\mathbf{t} \subset \mathbf{t}'$ if $\mathbf{t} \subseteq \mathbf{t}'$ and $\mathbf{t} \not\equiv \mathbf{t}'$.

Theorem 4.5. *Let* $\mathbf{t}, \mathbf{t}' \in \mathcal{T}^{exp}$. *Then* $\mathbf{t} \to_{ef} \mathbf{t}'$ *implies* $\mathbf{t} \subset \mathbf{t}'$.

Proof Obvious. □

Lemma 4.6. *Let* $\mathbf{t} \in \mathcal{T}^{exp}$ *and* $p\,n \in^{\wedge} \mathbf{t}$. *Assume that* n *is an* inner *num-variable. Then there is an* $\mathrm{L} \in p$ *that binds the* n *and a matching* $\mathrm{A} \in p$.

Proof An inner num-label n can only pop up in a $\to_{ef}$-reduction, when the L binding n is pivotal in the reduction and this L is matched to an A (see definition 4.3). This situation is maintained by the expanding nature of $\to_{ef}$-reduction (Theorem 4.5). □

4.2 Tracing the binder in expanding β-reduction

Let $\mathbf{t}_0 \in \mathcal{T}^{exp}$ and $\mathbf{t}_0 \twoheadrightarrow_{ef} \mathbf{t}$, so $\mathbf{t} \in \mathcal{T}^{exp}$ is the result of a series of expanding, focused reductions. These reductions may introduce inner variables, so it is not immediately clear what the binders are for (inner or outer) variables. In this section we investigate how one can determine the binder of a num-variable in $\mathbf{t}$.

Let $p\,n \in^{\wedge} \mathbf{t}$, so $p\,n$ is a root path. Here n can be an inner or an outer num-label. We describe a pushdown automaton $\mathcal{P}_{exp}$ that finds the L-*binder* of n, i.e., the label $\mathrm{L} \in p$ that binds n (this label always exists, since $\mathcal{T}^{exp}$ only contains closed terms).

We now present and explain the action of the pushdown automaton $\mathcal{P}_{exp}$. Let $\mathbf{t} \in \mathcal{T}^{exp}$ and $p\,n \in^{\wedge} \mathbf{t}$. Assume that we desire to apply algorithm $\mathcal{P}_{exp}$ to find the L-binder of n.

Remark 4.7. Preliminary remarks.

In 'algorithm' $\mathcal{P}_{exp}$, *we employ states that are pairs of natural numbers:* (k, l). *We start with the insertion of a pair* (n, i) *in the tail of the string* $p\,n$, *between* p *and* n. *Here* i *is originally either* 0 *or* 1. *The automaton moves the pair to the left through* p, *one step at a time, successively passing the labels in* p *and meanwhile adapting the numbers in the pair.*

The automaton has an outside stack that will contain certain states that are pushed on the top of the stack; a state on top of the stack can also be popped back, i.e., inserted into the path p, *again.*

The transitions are described in Definition 4.8. A possible one-step transition is denoted by the symbol $\to$ *(the reflexive, transitive closure of this relation is denoted* $\twoheadrightarrow$*).*

The procedure may be complicated by several recursive calls. In every recursive call, the algorithm starts with a 'new' num-variable j *in the path* p. *Subsequently, it not only finds the* L *binding this* j, *but also* the A matching the j.

The formal description of $\mathcal{P}_{exp}$ is the following.

Procedure 4.8. Preparation: Transform $p\,n$ into $p\,(n,i)\,n$, where $i = 0$ if the goal is to find the L-binder, and $i = 1$ in the recursion, when both the L-binder of n and the matching A are detected.

Now **start** $\mathcal{P}_{exp}$ employing the following transition rules.

(1) *first step: stack* $= \emptyset$
(2) p L (m,k) q $\rightarrow$ p $(m$–$1,k)$ L q, if $m > 0$
(3) p A (m,k) q $\rightarrow$ p (m,k) A q, if $m > 0$
(4) p S (m,k) q $\rightarrow$ p (m,k) S q, if $m > 0$
(5) p j (m,k) q $\rightarrow$ p $(j,1)$ j q, if $m > 0$; *push* (m,k)
(6) p L $(0,l)$ q $\rightarrow$ p $(0,l{+}1)$ L q, if $l > 0$
(7) p A $(0,l)$ q $\rightarrow$ p $(0,l$–$1)$ A q, if $l > 0$
(8) p j $(0,l)$ q $\rightarrow$ p $(0,l)$ j q, if $l > 0$
(9a) p $(0,0)$ q $\rightarrow$ p *pop* q, if *stack* $\neq \emptyset$
(9b) p $(0,0)$ q $\rightarrow$ *stop*, if *stack* $= \emptyset$.

Lemma 4.9. *Let* $\mathbf{t} \in \mathcal{T}^{exp}$ *and* $p\,n \in^{\wedge} \mathbf{t}$. *Apply* $\mathcal{P}_{exp}$ *to* $p\,(n,i)\,n$, *where* $i = 0$ *or* 1.

(*i*) *If* $\mathcal{P}_{exp}$ *stops in* $p'\,(0,0)\,q'\,n$ *with* empty *stack (see rule 9b), then each possible recursion has ended. Moreover,* $q'\,n \equiv \mathrm{L}\,q''\,n$ *and the mentioned* L *binds the* n.

(*ii*) *If* $\mathcal{P}_{exp}$ *stops in* $p'\,(0,0)\,q'\,n$ *with* non-empty *stack (see rule 9a), then the* A *matching the binding* L *of* n *has been found, the top-element of the stack is popped and* $\mathcal{P}_{exp}$ *continues where it had stopped before the recursion step.*

Proof The proof has been executed by one of the authors (Ferruccio Guidi), by means of the theorem prover Matita (Asperti *et al.*, 2011; Guidi, 2014).

5 Further Work and Acknowledgements

There is much more to be said about the notion of expanding β-reduction as presented in Section 4.1. We intend to do that in a forthcoming paper, including conventional proofs, theorems on expanding β-reduction and a connection between weak and strong normalization for this reduction.

I, Ferruccio Guidi, would like to dedicate the results presented in these pages to Anyelis Marielbys Parra, a special friend whose constant closeness accompanied me in the development of this work.

I, Rob Nederpelt, would express special thanks to Vincent van Oostrom for his interest in a pre-version of this paper. I also thank Herman Geuvers for encouraging remarks and a thorough review of an earlier version of this paper.

References

Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J.-J., Explicit Substitutions, *J. of Functional Programming*, Vol. 1, 375–416, Cambridge University Press, 1991.

Accattoli, B. and Kesner, D., The structural lambda-calculus. In Dawar, A. and Veith, H. eds, *CSL 2010*, Vol. 6247 of LNCS, 381-–395, Springer, 2010.

Accattoli, B. and Kesner, D., The permutative lambda calculus, *18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning - LPAR-18*, Merida, Venezuela, 2012.

Asperti, A., Ricciotti, W., Sacerdoti Coen, C. and Tassi, E., The Matita Interactive Theorem Prover. In Bjørner, N. and Sofronie-Stokkermans, V., eds, *Proceedings of the 23rd International Conference on Automated Deduction (CADE-2011)*, Vol. 6803 of LNCS, 64–69, Springer, 2011. X

Barenbaum, P. and Bonelli, E., Optimality and the Linear Substitution Calculus. In: Miller, D., ed., *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, 9:1—9:16, Leibniz International Proceedings in Informatics, 2017.

Barendregt, H.P., Lambda calculi with types. In Abramsky, S., Gabbay, D.M. and Maibaum, T., eds, *Handbook of Logic in Computer Science*, Vol. 2, 117–309, Oxford, 1992. X

de Bruijn, N.G., Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Indagationes Math.* 34 (1972), 381–392. Also in Nederpelt *et al.* (1994).

de Bruijn, N.G., *A namefree lambda calculus with formulas involving symbols that represent reference transforming mappings*, Eindhoven University of Technology, Dept. of Math., Memorandum 1977-10, 1977. (See also The Automath Archive AUT 050, www.win.tue.nl>Automath.)

de Bruijn, N.G. (1978a), *A namefree lambda calculus with facilities for internal definitions of expressions and segments*, Eindhoven University of Technology, EUT-report 78-WSK-03, 1978. (See also The Automath Archive AUT 059, www.win.tue.nl>Automath.)

de Bruijn, N.G. (1978b), Lambda calculus notation with namefree formulas involving symbols that represent reference transforming mappings, *Indagationes Math.* 81, 348–356, 1978. (See also The Automath Archive AUT 055, www.win.tue.nl>Automath.)

de Bruijn, N.G., Generalizing Automath by means of a lambda-typed lambda calculus, in: Kueker, D.W., Lopez-Escobar, E.K.G. and Smith, C.H., eds., *Mathematical Logic and Theoretical Computer Science*, New York, 1987.

Curien, P.-L., Hardin, Th. and Lévy, J.-J., Confluence Properties of Weak and Strong Calculi of Explicit Substitutions, *Journal of the ACM*, 43(2), 362–397, New York, 1996.

Guidi, F., *Landau's "Grundlagen der Analysis" from Automath to lambda-delta*, University of Bologna, Technical Report UBLCS 2009-16, 2009.

Guidi, F., *Formal specification for the interactive prover Matita 0.99.2*, 2014. (`http://lambdadelta.info/`)

Kamareddine, F. and Bloo, R., De Bruijn's syntax and reductional behaviour of lambda-terms: the typed case, *Journal of Logic and Algebraic Programming*, Vol. 62–1, p. 109–131, 2005.

Kamareddine, F.D. and Nederpelt, R.P., On stepwise explicit substitution, *Int. Journal of Foundations of Computer Science*, 4(3), 197–240, World Scientific Publishing Co, Singapore, 1993.

Kluge, W., Abstract Computing Machines — A Lambda Calculus Perspective, *Texts in Theoretical Computer Science*, EATCS Series, Springer-Verlag, 2005.

Nederpelt, R.P., *Strong normalisation in a typed lambda-calculus with lambda-structured types*, Ph.D. thesis, Eindhoven University of Technology, 1973. Also in Nederpelt *et al.* (1994).

Nederpelt, R.P., *A system of lambda-calculus possessing facilities for typing and abbreviating, Part I: Informal introduction*, Memorandum 1979-02, Department of Mathematics, Eindhoven University of Technology, The Automath Archive AUT 068, www.win.tue.nl>Automath, 1979.

Nederpelt, R.P., *A system of lambda-calculus possessing facilities for typing and abbreviating, Part II: Formal description*, Memorandum 1980-11, Department of Mathematics, Eindhoven University of Technology, The Automath Archive AUT 075, www.win.tue.nl>Automath, 1980.

Nederpelt, R.P., Geuvers, J.H. and de Vrijer, R.C., eds: *Selected Papers on Automath*, North-Holland, Elsevier, 1994.

Regnier, L., *Lambda-calcul et reseaux*, These de doctorat, Universite Paris 7, Janvier 1992.

Regnier, L., Une équivalence sur les lambda-termes, *Theoretical Computer Science 126(2)*, 281–292, 1994.

Rose, K.H., Explicit cyclic substitution, in: Rusinowitch, M., and Rémy, J.-L., eds., *Conditional Term Rewriting Systems*, Springer, Berlin, 36–50, 1993.