A semantics for a fine lambda-calculus with de Bruijn indices

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Semantics for a fine λ-calculus with de Bruijn indices

by

F. Kamareddine and R. Nederpelt

93/28

Computing Science Note 93/28
Eindhoven, September 1993

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

A Semantics for a fine λ-calculus with de Bruijn indices

Fairouz Kamareddine *
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@dcs.glasgow.ac.uk

and

Rob Nederpelt
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, the Netherlands
email: wsinrpn@win.tue.nl

September 7, 1993

*Kamareddine is grateful to the Department of Mathematics and Computing Science, Eindhoven University of Technology, for their financial support and hospitality from October 1991 to September 1992, and during the summer of 1993.

Name and mailing address of author to whom proofs should be sent:

Fairouz Kamareddine
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@dcs.glasgow.ac.uk

Contents

Abstract

Most of us who have worked with named variables in the λ-calculus must have noticed how sticky such variables can be. The problem is, that named variables play a very demanding role in the most basic operations of the λ-calculus, namely: β-reduction and substitution. This has lead to using implicit substitution rather than the explicit one in most theories of the λ-calculus. Variable names however, have one advantage that should not be underestimated; that is: they facilitate the readability of terms. Now, it would be very nice if we could write the basic operations of the λ-calculus in a precise way which avoids the messiness of variables. It would be very nice moreover, if we could sometimes keep the variable names, without having to pay the price usually associated with them. Our first task in this paper is to get rid of the problematic variable names and to establish what we believe is the most precise and fine λ-calculus, Ω_{Ξ}. In such a calculus, de Bruijn's indices are used instead of variable names and substitution and reduction are defined in a step-wise fashion which can be directly implemented without having to carry out a lot of book-keeping as is usually the case in the classical λ-calculus. Most importantly, the substitution in Ω_{Ξ} is no longer the implicit substitution but rather it is the explicit one which is long needed in many applications of the λ-calculus. Such an explicit substitution has been facilitated as a result of the fine structure of λ-terms that we propose in this paper and where *item notation* plays a dominant role. Furthermore, the species of variable names is cultivated and ordered so that a fine inter-marriage between de Bruijn's indices and variable names takes place. Such a relationship between de Bruijn's indices and variable names will be used to show the consistency of our fine reduction and explicit substitution in terms of the classical λ-calculus. We shall also reflect on the use and necessity of α-conversion.

Keywords: De Bruijn's indices, variable updating, substitution, reduction, soundness.

1 Introduction

We shall start in this paper by discussing a typed λ-calculus Λ which has the following features:

- There is no distinction between types and terms. This will make the calculus more general. See for example [Barendregt 91] and [Barendregt 92] where instead of terms and types, the notion of pseudo-terms is used. See furthermore [de Bruijn 70] where the Automath system provided is the most abstract formulation of type systems and where no distinction is made between types and terms. The selected papers in [NGdV 94] elaborate further on the Automath systems.

- The argument comes before the function so that instead of $(t_1 t_2)$ we write $(t_2 \delta t_1)$. This convention has a practical advantage which we will see below. In particular, it helps to show clearly which are the β-redexes.

- The type comes before the typed variable so that instead of $(\lambda_{v:t_1}.t_2)$ we write $(t_1 \lambda_v t_2)$. This convention is of less importance than the above convention but will play a role in providing a modular way of representing terms. That is, every non variable term can be looked at as an ω-item followed by a term, where the notion of ω-items for $\omega \in \{\lambda_v, \delta\}$, is explained below.

- The bracketing of the operators λ and δ are changed so that we write $(t_1 \lambda_v) t_2$ instead of $(t_1 \lambda_v t_2)$ and $(t_1 \delta) t_2$ instead of $(t_1 \delta t_2)$.

These conventions together, give rise to *items* like the λ-item $(t_1 \lambda_v)$ and the δ-item $(t_1 \delta)$. Moreover, the δ-item and the λ-item involved in a β-reduction occur *adjacently* in the term; they are not separated by the "body" of the term, that can be extremely long! This fashion of writing terms is close to the mathematical definitions and theorems as is elaborated in [Nederpelt 87]. In the system Λ, the usual implicit substitution of the λ-calculus is used.

The item notation enables us to add substitution items (or σ-items) which will have the same status as the λ- and δ-items hence making substitution an object level process and giving substitution items the right to be first-class citizens. In fact, thanks to the item notation we can provide the fine structure of the λ-calculus with various refined forms of reduction, substitution and term manipulation.

After presenting Λ, the calculus with item notation but where variable names and implicit substitution are used, we shall introduce a calculus based on Λ but where de Bruijn's indices and explicit substitution are used. For this, we start by introducing de Bruijn's indices. Such indices have the practical advantages that they avoid all the need to deal with variable renaming in terms (see [de Bruijn 72], [Abadi et al. 91], [CH 88] and [KN 93]). The calculus based on Λ and on de Bruijn's indices will be called Ω_Ξ for Ξ being the set of variables which are de Bruijn's indices together with ε a special variable. In the first instance, Ω is taken to be $\{\lambda, \delta\}$. In order to accommodate substitution explicitly and in order to discuss variable updating and term reduction, Ω is increased to $\{\lambda, \delta, \sigma, \varphi, \mu\}$. We add the σ-items for substitution, the φ-items for variable updating and the μ-items for β-reduction. The φ-items are written as $(\varphi^{(k,i)})$ for $i \geq 1$ and $k \geq 0$. The superscript k decides which variables are to be updated. The superscript i decides how much a variable must be updated; namely by increasing it by i. The σ-items are written as $(t\sigma^{(i)})$ for $i \geq 1$. $(t\sigma^{(i)})t'$ means: in t' substitute

t for i. The μ-items are written as $(\mu^{(i)})$ for $i \geq 1$. $(\mu^{(i)})t$ means, decrease all the variables in t that are $> i$ by 1.

We provide the φ-, σ-, μ- and β-reduction rules in Ω_{Ξ} which are all explicit and step-wise. Furthermore, these rules may be used to get local and global forms of reduction.

Ω_{Ξ} is the calculus of explicit substitution, which is based on what we call item notation and on the use of de Bruijn's indices. We provide a method which can take any term of Λ into Ω_{Ξ} such that all α-equivalent terms in Λ are mapped into a unique element of Ω_{Ξ}. The other direction however, of mapping elements of Ω_{Ξ} into elements of Λ is more difficult. This is because in Ω_{Ξ}, the λ's do not have variable names as subscripts and so we have to look for such subscripts in a way that no free variables in the term get bound. Now, the question that might be asked is why should we be interested in mapping elements of Ω_{Ξ} into Λ. After all, variable names in the λ-calculus are messy and the idea of the de Bruijn indices is to be precise and to avoid the clumsiness of variables. Moreover, a term in Ω_{Ξ} represents a whole class of terms in Λ; namely all those α-equivalent terms. So, in taking the term of Ω_{Ξ} back to Λ, which of these α-equivalent terms are we going to choose? Are we going to consider terms of Λ modulo α-conversion and then choose any term in the equivalence class? If so, then our work is pointless. In other words, what is the point of going from de Bruijn's indices to α-equivalence classes when de Bruijn indices actually represent the α-equivalence classes? Hence the first conclusion is that, in translating the terms from Ω_{Ξ} to Λ, we must avoid α-conversion in Λ and we must associate to each term of Ω_{Ξ} a unique term of Λ. This will also have advantages for implementation. For then, we know exactly which term we are working with. Now, having such a translation $|\cdot|$ from Ω_{Ξ} to Λ, our task is to show that the variable updating, the substitution and the reduction rules in Ω_{Ξ} are sound. We do this by showing that if $t \rightarrow t'$ where $\rightarrow$ is either σ-, or φ- or μ-reduction (excluding the σ- or the μ-generation and the σ-transition rules, see below), then $|t| \equiv |t'|$. That is, we show that all the rules which accommodate variable updating and substitution result in syntactically equal terms. We shall moreover, show that if $t \rightarrow t'$ where the reduction includes σ- or μ-generation, then $|t| =_{\overline{\alpha\beta}} |t'|$. That is, the rules which actually reduce β-redexes in Ω_{Ξ} are nothing more than the β rule in Λ. Finally if $\rightarrow$ is σ-transition then $|t| =_{\overline{\alpha}} |t'|$. These results are of course desirable, otherwise how can we check the correctness of our reduction rules. Furthermore, it should be noted that the semantics that we provide is a *flat* semantics. That is, the reduction steps in the fine λ-calculus are mapped to syntactical equality (except in the cases mentioned above), and not to a corresponding reduction. We provide the fine structure of the λ-calculus which has advantages that range over all areas and disciplines of λ-calculi and type theory, and we give a semantics which shows that our reduction and substitution rules are a refinement of those of the classical calculus.

We believe that our approach is the first to be so precise about variable manipulation, substitution and reduction in the λ-calculus. There is never a confusion of which variable is the one manipulated and hence a machine can easily carry out our reduction strategies and translate the terms using variables in a straightforward manner. We believe that the approach of this paper should be considered in implementations of functional languages and of theorem provers. Our work here might look too involved, but we have actually carried out the hard part of manipulating variables once and for all. No further work is needed afterwards on book-keeping of what happens to variables, terms or reductions either in proofs or in implementations. We are persuaded that this is the first precise formulation of λ-terms, variables and reductions. Furthermore, we believe that this formulation not only enables

explicit and local substitution as we show in this paper, but also enables a generalisation over all branches of λ-calculus and type theory (see [KN 93], [NK 9x] and [KN 9x]).

To sum up, we provide Λ, a calculus which uses item notation, variable names and explicit substitution. We extend Λ to Ω_{Ξ} where item notation is used with de Bruijn indices instead of variable names and explicit rather than implicit substitution. We provide the translation between both systems and in both directions. The translation from Ω_{Ξ} to Λ aims to show that our explicit and step-wise reduction and substitution rules are sound and are a refinment of the implicit rules of the λ-calculus. Furthermore, such a translation aims at furthering our understanding of when α-reduction is needed in the λ-calculus. In fact, we try to do completely without α-reduction until we are forced to use it. Moreover, this translation gives to every term with de Bruijn indices a unique term of Λ (with no mention of α-conversion).

2 The syntax of the calculi

2.1 The calculus Λ

We let V, the set of variables of Λ, be $\{\varepsilon\} \cup \mathcal{F}$, where $\mathcal{F} = \{x_1, x_2, \ldots\}$ and we take $v, v', v'', v_1, v_2, \ldots$ to range over $\mathcal{F}$. The variables $x_1, x_2, \ldots$ will be ordered as in Definition 2.16.

Notation 2.1 We take $\mathbb{N}$ to be the set of natural numbers, i.e. ≥ 0, $\mathbb{P}$ to be the set of positive natural numbers, i.e. > 0 and $\mathbb{Z}$ to be the set of integers.

Definition 2.2 *(Λ)*
We define Λ as follows:
$\Lambda ::= V \mid (\Lambda\lambda_{\mathcal{F}}\Lambda) \mid (\Lambda\delta\Lambda)$

We let $t, t_1, \ldots$ denote terms in Λ, and use $\omega, \omega', \omega_1, \ldots$ to range over the so-called *operators* $\{\delta\} \cup \{\lambda_v; v \in \mathcal{F}\}$. Moreover, ε is never used as a subscript for λ. The symbol ε can be looked at as a special variable or as a constant. It is added because it enables us to generalise the calculus. In fact, by taking all types of variables after λ to be ε, we obtain the type free λ-calculus. ε has further uses such as the $\Box$ in [Barendregt 91] (see [KN 93] and [NK 9x]).

The term $(t_1\lambda_v t_2)$ is to be understood as the classical λ-calculus term $(\lambda_{v:t_1}.t_2)$. The term $(t_1\delta t_2)$ is to be understood as the classical λ-calculus term $(t_2 t_1)$.

Notation 2.3 *(Item Notation)*
We shall place parentheses in Λ in an unorthodox manner: we write $(t_1\omega)t_2$ instead of $(t_1\omega t_2)$. The reason for using this format is, that both abstraction and application can be seen as the process of fixing a certain part (an "**item**") to a term:

- the abstraction $\lambda_{v:t'}.t$ is obtained by prefixing the abstraction-item $\lambda_{v:t'}$ to the term t. Hence, $(t'\lambda_v t)$ is obtained by prefixing $t'\lambda_v$ to t.
- the application tt' (in "classical" notation) is obtained by postfixing the argument-item t' to the term t. Now $(t'\delta t)$ is obtained by prefixing $t'\delta$ to t.

In item-notation we write in these cases $(t'\lambda_v)t$ and $(t'\delta)t$, respectively. Here both $(t'\lambda_v)$ and $(t'\delta)$ are *prefixed* to the term t. Moreover, in $(t\omega)$, if $t \equiv \varepsilon$ then it may be dropped. That is, we write (λ_v) instead of $(\varepsilon\lambda_v)$.

Definition 2.4 *(Items)*
If t is a term in item notation and ω is an operator, then $(t\omega)$ is an item. We use $s, s_1, s_i, \ldots$ as meta-variables for items.

Definition 2.5 *(Segments)*
A concatenation of zero or more items is a **segment**.

Notation 2.6 *(parentheses)*
Note the intended parsing convention:
In the term $(s_1 s_2 \ldots s_n v\omega)s'_1 s'_2 \ldots s'_m v'$, the operator ω combines the *full* term $s_1 s_2 \ldots s_n v$ with the *full* term $s'_1 s'_2 \ldots s'_m v'$.

Example 2.7 The term $(v\omega_1(v'\omega_2 v''))$ becomes in item-notation: $(v\omega_1)(v'\omega_2)v''$. Analogously, the term $((v\omega_2 v')\omega_1 v'')$ becomes $((v\omega_2)v'\omega_1)v''$.

Lemma 2.8 *Every term has the form $(t_1\omega_1)(t_2\omega_2)\ldots(t_n\omega_n)v$ for $t_1, t_2, \ldots, t_n$ terms, $\omega_1, \omega_2, \ldots, \omega_n$ operators, $n \geq 0$ and v a variable.*
Proof: *Easy.* □

Based on this lemma, we shall draw the tree of each term $(t_1\omega_1)(t_2\omega_2)\ldots(t_n\omega_n)v$ for $n \geq 1$ as follows: We position the root of the tree ω_1 in the lower left hand corner. We have chosen this manner of depicting a tree in order to maintain a close resemblance with the item notation of terms. This has also advantages in the sections to come. In fact, the item-notation suggests a partitioning of the term trees in vertical layers. For $(v'\omega_1)(v''\omega_2)v'''$, these layers are: the parts of the tree corresponding with $(v'\omega_1)$, $(v''\omega_2)$ and v'' (connected in the tree with two edges). For $((v'\omega_2)v''\omega_1)v'''$ these layers are: the part of the tree corresponding with $((v'\omega_2)v''\omega_1)$ and the one corresponding with v'''. Figure 1 is self explanatory.

$(v'\omega_1(v''\omega_2 v'''))$ $\quad$ $((v'\omega_2 v'')\omega_1 v''')$
$(v'\omega_1)(v''\omega_2)v'''$ $\quad$ $((v'\omega_2)v''\omega_1)v'''$

Figure 1: Layered trees, with normal layered notation and item-notation

Remark 2.9 Note that every term which is not a variable, has the form $(t\omega)t'$, from Definition 2.2 and Notation 2.3. Such a term is moreover, from Lemma 2.8, of the form $(t_1\omega_1)(t_2\omega_2)\ldots(t_n\omega_n)v$. Hence, $t \equiv t_1, \omega \equiv \omega_1$ and $t' \equiv (t_2\omega_2)\ldots(t_n\omega_n)v$.

Definition 2.10 *($FV(t)$, for $t \in \Lambda$)*

$$\begin{array}{lcll} FV(\varepsilon) & = & \emptyset & \\ FV(v) & = & \{v\} & \text{if } v \not\equiv \varepsilon \\ FV((t_1\lambda_v)t_2) & = & FV(t_1) \cup (FV(t_2) \setminus \{v\}) & \\ FV((t_1\delta)t_2) & = & FV(t_1) \cup FV(t_2) & \end{array}$$

Remark 2.11 Notice here that this definition might cause some confusion. For example take the term t to be $(v\lambda_v)v$, then $FV(t) = \{v\}$. In fact, $(v\lambda_v)v$ will be α-reducible to $(v\lambda_{v'})v'$ (see axiom (α) below). Such confusion will be avoided using de Bruijn's indices.

Definition 2.12 *($BV(t)$ for $t \in \Lambda$)*

$$\begin{array}{lcl} BV(v) & = & \emptyset \\ BV((t_1\lambda_v)t_2) & = & BV(t_1) \cup BV(t_2) \cup \{v\} \\ BV((t_1\delta)t_2) & = & BV(t_1) \cup BV(t_2) \end{array}$$

Note that ε is neither free nor bound.

Substitution in the λ-calculus is usually defined (up to some variation) as follows (see [Barendregt 84]):

Definition 2.13 *(Substitution in Λ)*
If t, t' are terms in Λ and v is a variable in V, we define the result of substituting t' for all the free occurrences of v in t as follows:

$$t[v := t'] =_{df} \begin{cases} t' & \text{if } t \equiv v \\ v' & \text{if } t \equiv v' \not\equiv v \\ (t_2[v := t']\delta)t_1[v := t'] & \text{if } t \equiv (t_2\delta)t_1 \\ (t_2[v := t']\lambda_v)t_1 & \text{if } t \equiv (t_2\lambda_v)t_1 \\ (t_2[v := t']\lambda_{v'})t_1[v := t'] & \text{if } t \equiv (t_2\lambda_{v'})t_1, v \not\equiv v', \\ & (v \notin FV(t_1) \text{ or } v' \notin FV(t')) \\ (t_2[v := t']\lambda_{v''})t_1[v' := v''][v := t'] & \text{if } t \equiv (t_2\lambda_{v'})t_1, v \not\equiv v', v \in FV(t_1), \\ & v' \in FV(t'), v'' \text{is the first variable} \\ & \text{in } \mathcal{F} \text{ which does not occur in } (t\delta)t' \end{cases}$$

The fundamental axioms of the λ-calculus are (α) and (β). Other axioms such as (η) (which is needed together with another axiom to derive extensionality) are optional. For this, we shall only concentrate on (α) and (β).

$$\begin{array}{ll} (\alpha) & (t\lambda_v)t' \to_\alpha (t\lambda_{v'})t'[v := v'] \text{ where } v' \notin FV(t') \\ (\beta) & (t''\delta)(t\lambda_v)t' \to_\beta t'[v := t''] \end{array}$$

Note that a so-called $\delta\lambda$-pair of items: $(t''\delta)(t\lambda_v)$, is a signal for a possible β-reduction. This $\delta\lambda$-pair *precedes* the term to which it applies.

We say that $t \to_\alpha t'$ (respectively $t \to_\beta t'$) just in case (α) (respectively (β)) takes t to t'. Moreover, we assume that $\to_\alpha$ and $\to_\beta$ are compatible where compatibility over T_λ is given by the following definition:

Definition 2.14 *(Compatibility over T_λ)*
We say that $\rightarrow_r$ where $r \in \{\alpha, \beta\}$ on T_λ is compatible if whenever $t \rightarrow_r t'$ we get: $tt_1 \rightarrow_r t't_1, t_1t \rightarrow_r t_1t', \lambda_{v:t}.t_1 \rightarrow_r \lambda_{v:t'}.t_1$ *and* $\lambda_{v:t_1}.t \rightarrow_r \lambda_{v:t_1}.t'$.

We call the reflexive transitive closure of $\rightarrow_\alpha$, $\twoheadrightarrow_\alpha$. Similarly $\twoheadrightarrow_\beta$ is the reflexive transitive closure of $\rightarrow_\beta$. We let $=_\alpha$ (respectively $=_\beta$) be the least equivalence relation closed under $\twoheadrightarrow_\alpha$ (respectively $\twoheadrightarrow_\beta$). Finally, $=$ is the least equivalence relation closed under $\twoheadrightarrow_\alpha$ and $\twoheadrightarrow_\beta$.

As obvious from our definition of substitution, we use $\equiv$ to be syntactic identity which accounts also for the parenthetes conventions. When $t = t'$ in Λ, we write $\vdash_\Lambda t = t'$.

2.2 De Bruijn's indices

De Bruijn in [de Bruijn 72] noted that due to the fact that terms as $\lambda_{x_1}.x_1$ and $\lambda_{x_2}.x_2$ are the "same" modulo α-conversion, one can find a λ-notation which expresses that similarity. That is, following de Bruijn, we can abandon variables and use indices instead. Examples 2.15, 2.17 below show how lambda terms can be denoted using de Bruijn's indices and example 2.18 illustrates how β-conversion works using such indices.

Example 2.15 Consider the type free lambda term $(\lambda_{x_1}.x_1)$. In this term, the x_1 following λ_{x_1} is a variable bound by this λ. In de Bruijn's notation, $\lambda_{x_1}.x_1$ and all its α-equivalent expressions can be written as $\lambda.1$. The bond between the bound variable x_1 and the operator λ is expressed by the number 1; the position of this number in the term is that of the bound variable x_1, and the value of the number ("*one*") tells us how many lambda's we have to count, going leftwards in the term, starting from the mentioned position, to find the binding place (in this case: the *first* λ to the left is the binding place).

De Bruijn's notation moreover, can be used for the typed λ-calculus. We illustrate here how the two terms $(\lambda_{x_3:x_2}.x_3)x_1$ and $\Lambda_A.\lambda_{x_1:A}.x_1$ can be represented using de Bruijn's indices. First, however, we need to account for the free variables x_1 and x_2. For this, we assume a free variable list:

Definition 2.16 *(Free variable list $\mathcal{F}$)*
For all terms, the free variable list is the same arbitrary but fixed, left-infinite list of λ_is with all i different variable names. Such a free variable list is called $\mathcal{F}$ and is given in Figure 2. Of course, for each term, having a finite number of free variables, a finite segment of this list suffices.

Example 2.17 The term $(\lambda_{x_3:x_2}.x_3)x_1$ is written as $(\lambda_2.1)1$. The free variables x_1 and x_2 in the typed lambda term are translated into the number 1 occurring after the term in parentheses, and the number 2: they refer to the "invisible" lambda's that are not present in the term, but may be thought of to *preceed* the term in the free variable list $\mathcal{F}$. In this example, the x_3 is *bound*, hence different from the *free* x_3 in $\mathcal{F}$. The bound x_3 is represented by the first number 1.
The term $\Lambda_A.\lambda_{x_1:A}.x_1$ can be represented by $\Lambda.\lambda_1.1$.

Some type theories insist on distinguishing λ and Π. The λ being used for the function and Π for the function *type*. Then the typed term $\Lambda_A.\Pi_{x_1:A}.x_1$ can be written as $\Lambda.\Pi_1.1$ where the 1 adjacent to Π, says that Λ is the binding operator for the type (viz. A) and the final 1 replaces the variable bound by Π.

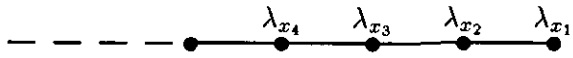


Figure 2: The free variable list $\mathcal{F}$

The described way of omitting binding variables, and rendering bound and free variables by means of so-called **reference numbers**, is precisely how de Bruijn's notation works. Next we see how β-reduction works in this notation.

Example 2.18 In ordinary lambda calculus, all the terms $(\lambda_{x_i:x_1}.(x_i x_3))x_2$, for $i \neq 3$, β-reduce to $x_2 x_3$, i.e. the result of substituting "argument" x_2 for x_i in $x_i x_3$. In de Bruijn's notation this becomes — under the assumption that the free variable list is $\lambda_{x_3}, \lambda_{x_2}, \lambda_{x_1}$: $(\lambda_1.1\ 4)2$ reduces to $2\ 3$. Here the contents of the subterm $1\ 4$ changes: 4 becomes 3. This is due to the fact that a λ-item, viz. (λ_1), disappeared (together with the argument 1). Furthermore, 1 changed to 2.

2.3 The syntax of Ω_Ξ

Now we shall take Λ but where de Bruijn's indices are used instead of variable names. That is, we will get rid of the variables in Λ and replace them by de Bruijn's indices. This would mean of course that we no longer would need each λ to carry the subscript x_i for $i \in I\!P$ or so on with it, but rather, the number would point to which λ binds which occurrence. The best way here is to start with an example.

Example 2.19 We take the term $t \equiv (x_1\delta)(x_2\lambda_{x_4})(x_3\delta)x_4$ whose tree is drawn in Figure 3. We need to remove x_4, x_3, x_2, x_1 and to replace them by numbers. For this, as we see that x_1, x_2, x_3 are free variables, we need to use the free variable list (see Figure 2). We append dashed lines to our tree in Figure 3 to show that λ's on the dashed lines are imaginary and not a part of the term (see Figure 4). Now for each variable, we draw thin lines ending in arrows, pointing at the λ binding the variable. These lines follow the path which leads from the variable to the root following the *left side* of the branches of the tree. In order to find the *index* replacing the variable name, we count the λ's on this path (not the δ's). For example, we draw the thin line going from x_4 following the path which leads from x_4 to the root, until we reach λ_{x_4}, the λ binding x_4. We end the arrow there and as we have only passed one λ, the x_4 should be replaced by 1. This is the only x_4 we have in the tree, and as there are no more

x_4's bound by this λ_{x_4}, we can safely remove the subscript x_4 from λ_{x_4}. For x_3, in drawing the thin line going from x_3 following the path which leads from x_3 to the root, keeping to the left side of the branches until we reach λ_{x_3}, we see that we pass four λs. Hence, the x_3 should be replaced by 4. Now replacing x_1 and x_2 will be left as exercises. Figure 4 is now self explanatory.

As in Example 2.17, the bound variable x_4 in t should not be confused with the free x_4 in the list $\mathcal{F}$.

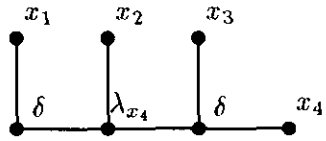
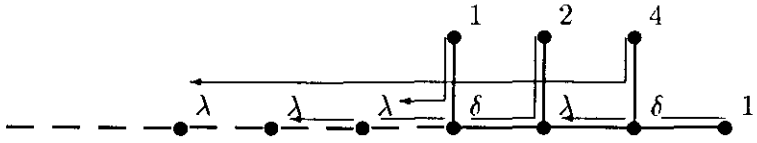


Figure 3: The tree of $(x_1\delta)(x_2\lambda_{x_4})(x_3\delta)x_4$



$(\lambda_{x_4:x_2}.x_4x_3)x_1$

$(x_1\delta)(x_2\lambda_{x_4})(x_3\delta)x_4$

$(1\delta)(2\lambda)(4\delta)1$

Figure 4: A tree with de Bruijn's indices

Note that we get the same de Bruijn trees for all terms $(x_1\delta)(x_2\lambda_{x_i})(x_3\delta)x_i$ for $i \neq 3, i \in I\!P$. This is due to the fact that de Bruijn's indices give the terms modulo α-conversion. In the case $i = 1$, or $i = 2$, we have here that x occurs both bound *and* free. These occurrences should be separated, as-is actually the case in the version with de Bruijn's indices. In order to translate $(x_1\delta)(x_2\lambda_{x_i})(x_3\delta)x_i$ for the case where $i = 1$ or $i = 2$, we have to rename x_i to x_j for $j > 3$.

Definition 2.20 *(Variables)*
As we decided to use indices instead of variables, we take Ξ the set of **variables** to be $\Xi = \{\varepsilon, 1, 2, \ldots\}$. Sometimes we will need to use actual variables, but this is not a part of our syntax. It is only a matter of simplifying the conversation. We use $i, j, m, n, \ldots$ to denote elements of $\{1, 2, \ldots\}$.

Using $\Omega = \{\delta, \lambda\}$ and Ξ we define our terms to be those symbol strings obtained in the usual manner on the basis of Ξ, the operators in Ω and parentheses. That is: Ω_Ξ is the free Ω-structure generated by Ξ.

Definition 2.21 *(Ω_Ξ)*
We define Ω_Ξ as follows:

$$\Omega_\Xi ::= \Xi \mid (\Omega_\Xi \lambda \Omega_\Xi) \mid (\Omega_\Xi \delta \Omega_\Xi)$$

As in Λ, we take $t, t_1, \ldots$ to denote terms in Ω_Ξ. We call the terms of Ω_Ξ in case $\Omega = \{\lambda, \delta\}$, $\Omega_{\lambda\delta}$-**terms** or simply **terms**. Later on we will increase Ω by adding σ, φ and μ. μ-terms will only be used with $\Omega_{\lambda\delta}$-terms. An important class of terms however is the $\Omega_{\lambda\delta\sigma\varphi}$-terms.

Now we take the same notational conventions as those for Λ given in Notations 2.3 and 2.6, and we define items and segments similarly. We take $\omega, \omega', \omega_1, \omega_2 \ldots$ to range over Ω. In the rest of this paper, we write terms of Λ and Ω_Ξ using the item notation.

Simple examples of terms are: ε, 3, $(2\delta)(\varepsilon\lambda)1$. Example 2.22 shows terms represented in Λ and Ω_Ξ. The translation function between Λ and Ω_Ξ will be given in the following section.

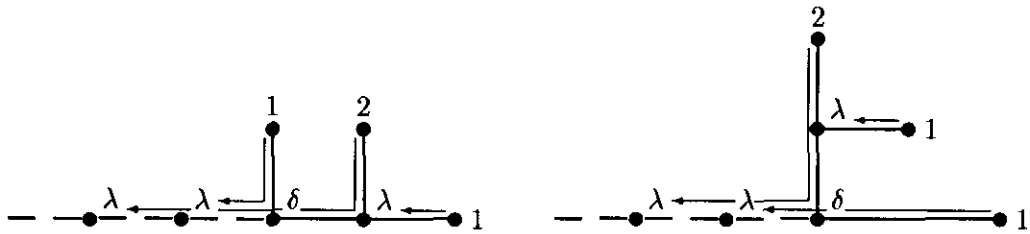
Example 2.22

- Consider the typed lambda term $(x_1\delta)(x_2\lambda_{x_5})x_5$. In Ω_Ξ, it is denoted as $(1\delta)(2\lambda)1$. The typed lambda term $(x_1\delta)(x_2\lambda_{x_3})x_3$ has the same denotation in Ω_Ξ. Note however, that $(x_1\delta)(x_2\lambda_{x_5})x_5 \not\equiv (x_1\delta)(x_2\lambda_{x_3})x_3$ for example, unless (α) is assumed in Λ.
- The typed lambda term $((x_2\lambda_{x_5})x_5\delta)x_1$ in Λ is written as $((2\lambda)1\delta)1$ in Ω_Ξ.
- The de Bruijn trees of these lambda terms are given in Figure 5.

Finally, we define a number of concepts connected with terms, items and segments. These will be used in the rest of the paper.

Definition 2.23 *(main items, main segments, ω-items, $\omega_1 \ldots \omega_n$-segments, body, weight)*

- *Each term t is the concatenation of zero or more items and a variable: $t \equiv s_1 s_2 \ldots s_n v$. These items $s_1, s_2, \ldots, s_n$ are called the* **main items** *of t.*
- *Analogously, a segment $\overline{s}$ is a concatenation of zero or more items: $\overline{s} \equiv s_1 s_2 \ldots s_n$; again, these items $s_1, s_2, \ldots, s_n$ (if any) are called the* **main items**, *this time of $\overline{s}$.*
- *A concatenation of adjacent main items (in t or $\overline{s}$), $s_m \ldots s_{m+k}$, is called a* **main segment** *(in t or $\overline{s}$).*



$(1\delta)(2\lambda)1$
$(x_1\delta)(x_2\lambda_{x_5})x_5$
$(\lambda_{x_5:x_2} \,.\, x_5)x_1$

$((2\lambda)1\ \delta)1$
$((x_3\lambda_{x_5})x_5\ \delta)x_1$
$x_1(\lambda_{x_5:x_3} \,.\, x_5)$

Figure 5: de Bruijn trees with explicit free variable lists and reference numbers

- *An item $(t\ \omega)$ is called an* ω-**item**. *Hence, we may speak about* λ-**items**, δ-**items** *(and later on about* σ-**items** *and* φ-**items**).
- *If a segment consists of a concatenation of an ω_1-item up to an ω_n-item, this segment may be referred to as being an* $\omega_1 \ldots \omega_n$-**segment**.
- *An important case of a segment is that of a* $\delta\lambda$-**segment**, *being a δ-item immediately followed by a λ-item.*
- *If $t \equiv \overline{s}v$, then $\overline{s}$ is called the* **body** *of t.*
- *The* **weight** *of a segment is the number of its main items.*

Example 2.24 Let the term t be defined as $(\varepsilon\lambda)((1\delta)(\varepsilon\lambda)1\delta)(2\lambda)1$ and let the segment $\overline{s}$ be $(\varepsilon\lambda)((1\delta)(\varepsilon\lambda)1\delta)(2\lambda)$. Then the main items of both t and $\overline{s}$ are $(\varepsilon\lambda)$, $((1\delta)(\varepsilon\lambda)1\delta)$ and (2λ), being a λ-item, a δ-item, and another λ-item. Moreover, $((1\delta)(\varepsilon\lambda)1\delta)(2\lambda)$ is an example of a main segment of both t and $\overline{s}$, which is a $\delta\lambda$-segment. Also, $\overline{s}$ is a $\lambda\delta\lambda$-segment, which is a main segment of t.

Now we define nl which counts the number of λ's in a term.

Definition 2.25 *(nl)*

$$\begin{array}{lll} nl(\varepsilon) & =_{df} & \emptyset \\ nl((t_1\delta)t_2 & =_{df} & nl(t_1) + nl(t_2) \\ nl((t_1\lambda)t_2) & =_{df} & nl(t_1) + 1 + nl(t_2) \end{array}$$

Note that $weight(t)$ is not necessarily the same as $nl(t)$. For example, $weight((1\lambda)2\lambda)3) = 1$ whereas $nl(((1\lambda)2\lambda)3) = 2$.

3 Axioms of Ω_Ξ

α-reduction is not needed for Ω_Ξ, precisely because we no longer have variables (de Bruijn's indices got rid of them). So now, we no longer have different ways of writing the same term

as we have taken the equivalence classes so that $\lambda_{x_1:x_3}.x_1, \lambda_{x_2:x_3}.x_2, \ldots$ all are represented by $(3\lambda)1$. For β-reduction, this is a bit more complicated. Let us start by an informal example, but the mechanical procedure will be given below:

Example 3.1 Now for β-reduction, the term $(x_1\delta)(x_2\lambda_{x_4})(x_3\delta)x_4$ reduces to $(x_3\delta)x_1$ (see Example 2.18 and Figure 6). Note that the presence of a so-called $\delta\lambda$-segment (i.e. a δ-item immediately followed by a λ-item), in this example: $(x_1\delta)(x_2\lambda_{x_4})$, is the signal for a possible β-reduction. Using de Bruijn's indices, this becomes (remember that the free variable list ends in $\lambda_{x_3}, \lambda_{x_2}, \lambda_{x_1}$): $(1\delta)(2\lambda)(4\delta)1$ reduces to $(3\delta)1$. In fact, if you look at Figure 6, you see that what is happening is that the $\delta\lambda$-segment $(1\delta)(2\lambda)$ has been cut off the tree, and the remaining term to the right of this segment has shifted to the left so that its root (i.e. the root of its tree) will occupy the place where the δ of $(1\delta)(2\lambda)$ used to be. That is not all of course. The 4 has to be decreased to 3 as we have lost one λ. The 1 in $(4\delta)1$ has to be replaced by the 1 of (1δ). The result is hence $(3\delta)1$.

The process could hence be summarised by saying that when contracting the redex $(t_1\delta)(t_2\lambda)$ in $(t_1\delta)(t_2\lambda)t$, all free variables in t must be decreased by 1 and all variables in t that are bound by the λ of $(t_2\lambda)$ must be replaced by t_1. This might be tricky however, for assume we write

$$(t_1\delta)(t_2\lambda)t \to_\beta t[1 := t_1, 2 := 1, 3 := 2, \ldots]$$

where $t[1 := t_1, 2 := 1, 3 := 2, \ldots]$ stands for the term t with 1 replaced by t_1, 2 replaced by 1 and so on. This substitution is moreover simultaneous. Now, assume furthermore that t is of the form $(\varepsilon\lambda)t'$. Then for the substitution $t[1 := t_1, 2 := 1, 3 := 2, \ldots]$ we must perform $((\varepsilon\lambda)t')[1 := t_1, 2 := 1, 3 := 2, \ldots]$.

Now, replacing $((\varepsilon\lambda)t')[1 := t', 2 := 1, 3 := 2, \ldots]$ by $(\varepsilon\lambda)t'[1 := t_1, 2 := 1, 3 := 2, \ldots]$ would not work. Rather it should be:

$$(\varepsilon\lambda)t'[1 := 1, 2 := t_1[1 := 2, 2 := 3, \ldots], 3 := 2, \ldots].$$

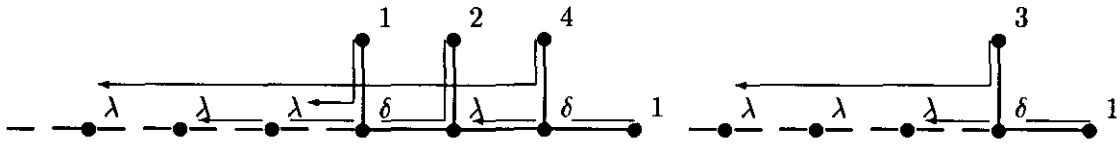
Based on this observation, we need to increment variables correctly in a term. Therefore we introduce an updating procedure which we call φ-reduction.

3.1 φ-reduction

Updating variables in a term will take place for example when a term t' is to be substituted for one or many occurrences of a variable v in a term t. What will then happen is that t' cannot be just thrown in t at the targeted occurrences of v, because t may have many λ's to the left of the targeted occurrence of v. This means that t' must be updated to take into account these extra λ's. The following example illustrates the point.

Example 3.2 Let $t \equiv (2\lambda)2$ and let $t' \equiv 3$. Now, if we want to replace the second occurrence of 2 in t by t', we cannot just remove 2 and replace it by 3. If we do so, we would obtain $(2\lambda)3$ which is not at all the result of the substitution. The result of the substitution should be $(2\lambda)4$. The idea is that, in replacing the second 2 in $(2\lambda)2$ by 3, the 3 has to be increased by 1, as it is now in the scope of one extra λ.

In order to update variables in a term, we add a new kind of items, **φ-items**. Let us for now assume that we write $(\varphi)t$ to increase the variables of t by 1. So in the above example, when



$(\lambda_{x_i:x_2}.x_i x_3)x_1$
$(x_1\delta)(x_2\lambda_{x_i})(x_3\delta)x_i$
$(1\delta)(2\lambda)(4\delta)1$

$x_1 x_3$
$(x_3\delta)x_1$
$(3\delta)1$

Figure 6: β-reduction in our notation

replacing the second 2 in $(2\lambda)2$ by 3, we really want to obtain $(2\lambda)(\varphi)3$. The process however is not that simple. Assume we want to replace 2° (where $^\circ$ points to the particular occurrence of 2) in $(2\lambda)2^\circ$ by $(\varepsilon\lambda)(1\delta)2$. Then, what is the result of $(2\lambda)(\varphi)(\varepsilon\lambda)(1\delta)2$? Which variables in $(\varepsilon\lambda)(1\delta)2$ have to be increased? Of course ε remains untouched. 1 moreover must remain untouched, as it is connected to the λ in $(\varepsilon\lambda)$. Hence it is only the 2 of $(\varepsilon)(1\delta)2$ which should be increased to 3. So how do we (in a step-wise fashion) decide which variables in a term are to be increased and which are not?

Note that all those variables of $(\varepsilon\lambda)(1\delta)2$ that have to be updated are *free* variables. Let us hence index φ. That is, we use φ as a (unary prefix) function symbol $\varphi^{(k,i)}$ with two parameters $k \geq 0$ and $i \geq 1$. The intention of the superscripts when $(\varphi^{(k,i)})$ travels through t_1 is the following:

- Superscript i preserves the **increment** desired for the free variables in t_1. This superscript does not increase when passing other λ's.
- Superscript k counts the λ's that are internally passed by in t_1 (k = '**threshold**'). This Superscript increases when passing another λ. The idea is that only the variables greater than k have to be increased, as those variables $\leq k$ are bound and hence should not be increased.

The effect of the updating must be that all free variables in t_1 increase with an amount of i; the k is meant to identify the free variables in t_1.

Note that the body of a φ-item is always the empty term.

Now of course updating variables by looking at the tree is an easy process. Just check how many λ's you have gone through before a free variable and increase the free variable by the number of λ's passed. This should happen for all variables in a term. This is achieved by letting the φ-items propagate upwards and to the right of the tree scanned. The following example illustrates the point:

Example 3.3 Assume you want to replace in $(\varepsilon\lambda)(2\lambda)3$, the 2 and the 3 by $(\varepsilon\lambda)2$. Then the result should be $(\varepsilon\lambda)((\varepsilon\lambda)3\lambda)(\varepsilon\lambda)4$. I.e. the 2 has been replaced by $(\varepsilon\lambda)3$ (due to the one extra λ that is now before $(\varepsilon\lambda)2$) and the 3 has been replaced by $(\varepsilon\lambda)4$ (due to the two extra λ's that are now before $(\varepsilon\lambda)2$). Figure 7 is self explanatory.

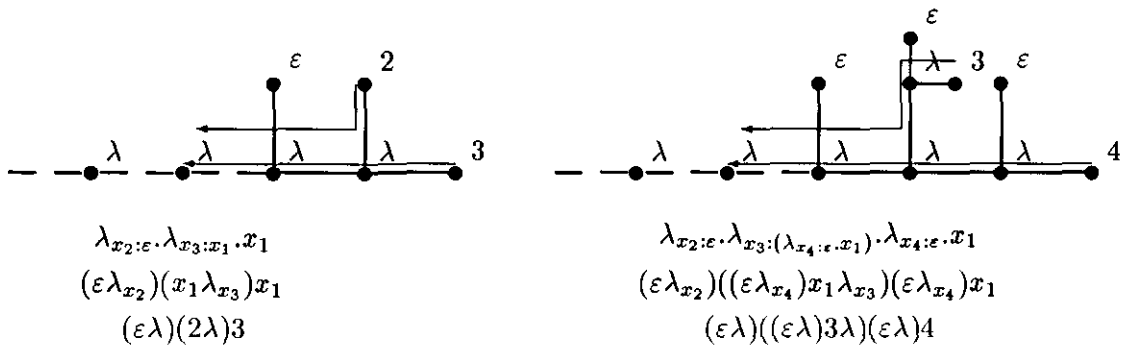


Figure 7: Substitution in our notation

The definition below formalises the updating process.

Definition 3.4 *(φ-reduction)*
For $k \in I\!N, i \in I\!P, v \in \Xi$ and t an $\Omega_{\lambda\delta}$-term, we have:
(φ-transition rules:)

$$\begin{array}{lll} (\varphi^{(k,i)})(t\lambda) & \rightarrow_\varphi & ((\varphi^{(k,i)})t\lambda)(\varphi^{(k+1,i)}) \\ (\varphi^{(k,i)})(t\delta) & \rightarrow_\varphi & ((\varphi^{(k,i)})t\delta)(\varphi^{(k,i)}) \end{array}$$

(φ-destruction rules:)

$$\begin{array}{lll} (\varphi^{(k,i)})v & \rightarrow_\varphi & v+i \quad \text{if } v > k \\ (\varphi^{(k,i)})v & \rightarrow_\varphi & v \quad\quad \text{if } v \leq k \text{ or } v \equiv \varepsilon \end{array}$$

The following details about these rules are to be noted.

- A term of the form $(\varphi^{(k,i)})t$ will be either such that t is a variable or a λ-item or a δ-item. In the case t is a variable, we use the φ-destruction rule. In the case of a δ-item or a λ-item, we have to update all the variables so that we keep the right references.
- The case where $(\varphi^{(k,i)})$ is to the left of a variable, we use one of two φ-destruction rules, the first for the case that v is free in the original t_1 (then a real update occurs), the second for the case that v is bound in t_1 or $v \equiv \varepsilon$ (then nothing happens with v).

Remark 3.5 Note that we introduce $\rightarrow_\varphi$ as a relation between segments, although it is meant to be a relation between terms. The rules must be read as follows: rule $\overline{s} \rightarrow_\varphi \overline{s'}$ states that $t \rightarrow_\varphi t'$ when a segment of the form $\overline{s}$ occurs in t, where t' is the result of the replacement of this $\overline{s}$ by $\overline{s'}$ in t. In other words, we implicitly assume compatibility (see [Barendregt 92]).

We denote the reflexive and transitive closure of $\rightarrow_\varphi$ by $\twoheadrightarrow_\varphi$.

Example 3.6 In substituting $(\varepsilon\lambda)2$ for 2 in $(\varepsilon\lambda)(2\lambda)3$, we have to compensate for one extra λ: the one preceding the 2 in $(\varepsilon\lambda)(2\lambda)3$. This can be done by substituting $(\varphi^{(0,1)})(\varepsilon\lambda)2$ for this 2. Then:

$(\varepsilon\lambda)((\varphi^{(0,1)})(\varepsilon\lambda)2\lambda)3 \rightarrow_{\varphi}$
$(\varepsilon\lambda)(((\varphi^{(0,1)})\varepsilon\lambda)(\varphi^{(1,1)})2\lambda)3 \twoheadrightarrow_{\varphi}$
$(\varepsilon\lambda)((\varepsilon\lambda)3\lambda)3$

Similarly, in the substitution of $(\varepsilon\lambda)2$ for 3 in $(\varepsilon\lambda)(2\lambda)3$, we have to compensate for two extra λs:

$(\varepsilon\lambda)(2\lambda)(\varphi^{(0,2)})(\varepsilon\lambda)2 \twoheadrightarrow_{\varphi} (\varepsilon\lambda)(2\lambda)(\varepsilon\lambda)4.$

Note that φ can be used to *increase* certain reference numbers. There is a case, however, when we wish to *decrease* a reference number: when we remove the $\delta\lambda$-segment in a β-reduction, the variables in the remaining part of the term in which β-reduction took place, must be decreased by 1, because one λ has disappeared. We will come back to this matter in Definition 3.14.

For convenience sake, we may drop the first superscript or both superscripts of the φ, according to the following definition:

Definition 3.7 *(φ-abbreviation)*
For all $i \geq 1$, $\varphi^{(i)}$ denotes $\varphi^{(0,i)}$. Moreover, φ denotes $\varphi^{(1)}$ (hence $= \varphi^{(0,1)}$).

3.2 σ-reduction

In order to be able to push substitutions ahead, step by step, we shall introduce a new kind of items, called **substitution items** (or σ-items). These σ-items can move through the branches of the term, step-wise, from one node to an adjacent one, until they reach a leaf of the tree. At the leaf, if appropriate, a σ-item can cause the desired substitution effect.

In this manner these substitution items can bring about different kinds of β-reductions. Note that we have chosen to make substitution a part of the formal language for the terms; we do not treat it as a meta-operation, as is usually done.

We use σ as an *indexed* operator, numbered with superscripts: $\sigma^{(1)}, \sigma^{(2)}, \ldots$. Hence, a σ-item has the form: $(t'\sigma^{(i)})$.

The notions: term, item, segment etc. take the extended $\Omega = \{\lambda, \delta, \sigma, \varphi\}$ into account.

Our terms now are $\Omega_{\lambda\delta\sigma\varphi}$-terms.

The intended meaning of a σ-item $(t'\sigma^{(i)})$ is: term t' is a candidate to be substituted for one or more occurrence of a certain variable; the superscript i selects the appropriate occurrences.

Now we can give the rules for *one-step σ-reduction*. This relation is denoted by the symbol $\rightarrow_{\sigma}$. The relation *σ-reduction* is the reflexive and transitive closure of one-step substitution. It is denoted by $\twoheadrightarrow_{\sigma}$. Similarly to our remark about φ in Remark 3.5, we introduce $\rightarrow_{\sigma}$ as a relation between segments, although it is meant to be a relation between terms. The rules must be read as follows: rule $\overline{s} \rightarrow_{\sigma} \overline{s'}$ states that $t \rightarrow_{\sigma} t'$ when a segment of the form $\overline{s}$ occurs in t, where t' is the result of the replacement of this $\overline{s}$ by $\overline{s'}$ in t.

We keep to the same meta-level notation as before, but let $\omega, \omega_1, \omega_2, \ldots$ range over λ, δ, φ and σ.

Now, in order to keep the references inside a σ-item correct during the process of σ-transition, a φ-item $(\varphi^{(k,i)})$ with $k = 0$ and $i = 1$ is added inside the σ-item, as follows: $((\varphi)t\sigma^{(j)})$. Here are the σ-reduction rules:

Definition 3.8 *(one-step σ-reduction)*
For $i \in I\!P, v \in \Xi, t_1, t_2\ \Omega_{\lambda\delta}$-terms, we have:
(σ-generation rule:)

$$(t_1\delta)(t_2\lambda) \rightarrow_\sigma (t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})$$

(σ-transition rules:)

$$\begin{array}{lll} (t_1\sigma^{(i)})(t_2\lambda) & \rightarrow_\sigma & ((t_1\sigma^{(i)})t_2\lambda)((\varphi)t_1\sigma^{(i+1)}) \\ (t_1\sigma^{(i)})(t_2\delta) & \rightarrow_\sigma & ((t_1\sigma^{(i)})t_2\delta)(t_1\sigma^{(i)}) \end{array}$$

(σ-destruction rules:)

$$\begin{array}{lll} (t_1\sigma^{(i)})i & \rightarrow_\sigma & t_1 \\ (t_1\sigma^{(i)})v & \rightarrow_\sigma & v \text{ if } v \not\equiv i \end{array}$$

Note that in the σ-transition rules, when a σ-item jumps over a λ-item, then the superscript of the σ increases by one. This is because that superscript counts the number of λ's actually passed, in order to find the right (occurrence of the) variable involved.

The σ-destruction rules apply when the σ-item has reached a leaf of the tree. When the superscript i of the σ is in accordance with the value of the variable, then we have met an intended occurrence of the variable; the substitution of t_1 for i takes place. When the superscript of σ and the variable in question do not match, then nothing happens to the variable, and the σ-item vanishes without effect.

Finally, we note that our transition rules as given here do not allow for σ-items to "pass" other σ-items.

Compare the σ-generation rule with (β) as defined in Section 2. Our rule, does not get rid of $(t_1\delta)(t_2\lambda)$ but keeps it because we may allow for local β-reduction by changing the σ-transition rules so some variables will still be bound by the λ in $(t_2\lambda)$. We shall see in Definition 3.14 how we can dispose of a reducible segment when there are no more customers for the λ involved, i.e. when there is no variable bound by this λ in the term.

The following lemma shows that σ-reduction reaches eventually all occurrences to be substituted. I.e., there is a path for global β-reduction.

Lemma 3.9 *In $(t_1\delta)(t_2\lambda)t_3$, σ-reduction substitutes t_1 for all occurrences of the variables bound by the λ of $(t_2\lambda)$ in t_3.*
Proof: *The proof is by an easy induction on t_3 in $(t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})t_3$.* □

Lemma 3.10 *In $(t_1\sigma^{(i)})t_2$, σ-reduction substitutes t_1 for all occurrences of variables in t_2 which are bound by the same λ being the i-th entry (from the right) in the free variable list of t_2. Moreover, the (φ)'s look after the updating of t_2.*
Proof: *By induction on t_2, noting that during propagation, everytime the σ-item passes a λ, the superscript at the top of σ is increased by 1. Hence keeping track of the variable to be substituted for.* □

The example below demonstrates how σ-reduction works.

Example 3.11

1. $(2\sigma^{(1)})(4\delta)1 \rightarrow_\sigma ((2\sigma^{(1)})4\delta)(2\sigma^{(1)})1 \twoheadrightarrow_\sigma (4\delta)2.$
2. $((3\delta)2\sigma^{(1)})(1\lambda)1 \rightarrow_\sigma (((3\delta)2\sigma^{(1)})1\lambda)((\varphi)(3\delta)2\sigma^{(2)})1 \rightarrow_{\sigma\varphi} ((3\delta)2\lambda)((4\delta)3\sigma^{(2)})1 \rightarrow_\sigma ((3\delta)2\lambda)1.$
3. $((3\delta)2\sigma^{(4)})(1\lambda)1 \rightarrow_\sigma (((3\delta)2\sigma^{(4)})1\lambda)((\varphi)(3\delta)2\sigma^{(5)})1 \rightarrow_{\sigma\varphi} (1\lambda)((4\delta)3\sigma^{(5)})1 \rightarrow_\sigma (1\lambda)1.$
4. $(1\delta)(2\lambda)(3\lambda)2 \rightarrow_\sigma$
   $(1\delta)(2\lambda)((\varphi)1\ \sigma^{(1)})(3\lambda)2 \rightarrow_\sigma$
   $(1\delta)(2\lambda)(((\varphi)1\ \sigma^{(1)})3\lambda)((\varphi)(\varphi)1\ \sigma^{(2)})2 \twoheadrightarrow_{\sigma,\varphi}$
   $(1\delta)(2\lambda)(((\varphi)1\ \sigma^{(1)})3\lambda)3 \rightarrow_\sigma$
   $(1\delta)(2\lambda)(3\lambda)3$

Now the following lemma shows that the right bond between variables and their binding λ's are maintained.

Lemma 3.12 *If $\overline{s}(t_1\delta)(t_2\lambda)t \rightarrow_\sigma \overline{s}(t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})t$ then in $\overline{s}(t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})t$, all variable occurrences are bound by the same λ's which bound them in $\overline{s}(t_1\delta)(t_2\lambda)t$.*

Proof: *We will only show how some cases can be carried out. The rest will be an easy exercise left to the reader. Let x be a variable in $(t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})$. There are only two cases to consider.*

- *case v occurs in $(t_1\delta)(t_2\lambda)$, then nothing to prove, as nothing has changed for that occurrence.*
- *case v occurs in $((\varphi^{(0,1)})t_1\sigma^{(1)})$, in particular in t_1 then a bound variable in t_1 clearly remains bound by the same λ in t_1. A free variable v in t_1 becomes updated by 1 by the $\varphi^{(0,1)}$. This is exactly what is intended, since there is one extra λ that v has to go through on its way to its λ. That is, the λ of $(t_2\lambda)$.*

□

Finally, we shall not discuss local substitution (the reader is referred to [KN 93]). We shall however just mention that by adding the σ-destruction rule:

$$(t_1\sigma^{(i)})t \rightarrow_\sigma t$$

to Definition 3.8, local substitution becomes available in the system. The reader is invited to check this.

3.3 β-reduction

Now let us consider β-reduction. Recall that in σ-generation, we generated σ-items. This will be repeated below:

Definition 3.13 *(σ-generation repeated)*
$(t_1\delta)(t_2\lambda) \rightarrow_\sigma (t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})$.

Recall that the (φ) is meant to compensate for the "extra" λ being passed. Recall moreover that the $\delta\lambda$-pair $(t_1\delta)(t_2\lambda)$ is *not* omitted. This is because we may want *local* substitution only.

Now, the reducible segment may be "without customers". Then σ-generation is undesirable since this leads to useless efforts. Hence it seems a wise policy to restrict the use of the σ-generation rule to those cases where the main λ of the reducible segment does actually bind at least one variable. When this is *not* the case, we shall speak of a **void $\delta\lambda$-segment.** Such a segment may be removed. One may compare this case to the application of a constant function to some argument; the result is always the (unchanged) body of the function in question. In this section, we shall present two ways of removing void $\delta\lambda$-segments.

3.3.1 Making i negative in $(\varphi^{(k,i)})$

Up to now, the i-superscript in $(\varphi^{(k,i)})$ has been considered an element of $I\!P$. If however, we allow in $(\varphi^{(k,i)})$, i to be negative, we could include the following rule:

Definition 3.14 *($\delta\lambda$-destruction rule)*
For all $t_1, t_2\, \Omega_{\lambda\delta}$-terms, we have: $(t_1\delta)(t_2\lambda) \to_\emptyset (\varphi^{(0,-1)})$ provided that the λ in $(t_2\lambda)$ does not bind any variable in the term following $(t_1\delta)(t_2\lambda)$, i.e. provided that $(t_1\delta)(t_2\lambda)$ is void.
Sometimes we denote $\to_\emptyset$ by **void β-reduction.**

It is clear that the provision in this definition is necessary: otherwise, bound variables would become, unintentionally, free. The updating $(\varphi^{(0,-1)})$-item is meant to compensate for the disappearing λ. Now, even though the superscript -1 is negative; this does not cause problems, precisely since the λ of $(t_2\lambda)$ does not bind any variable in the term following it. In fact, negative superscripts can have the effect that *different* variables become identified:

$$(\varphi^{(1,-1)})(2\delta)1 \twoheadrightarrow_\varphi (1\delta)1$$

Hence, updating is no longer an injection, which can be highly undesirable.

We note, however, that the mentioned unpleasant effects do not occur in the setting presented above: a φ-item with a negative exponent only occurs after the clean-up of a void $\delta\lambda$-segment, hence with a λ that does not bind any variable. Therefore, the injective property of updating is not threatened.

Now the σ-rules together with the $\delta\lambda$-destruction rule, enable us to accomplish (the usual) β-reduction as a combination of σ-steps and φ-steps:

Definition 3.15 *(one-step β-reduction $\to_{\beta'}$)*
One-step β-reduction of an $\Omega_{\lambda\delta}$-term is the combination of one σ-generation from a $\delta\lambda$-segment $\overline{s}$, the transition of the generated σ-item through the appropriate subterm in a global manner, followed by a number of σ-destructions, and updated by φ-items until again an $\Omega_{\lambda\delta}$-term is obtained. Finally, there follows one void β-reduction (i.e. a $\delta\lambda$-destruction) for the disposal of $\overline{s}$, and we use the φ-rules to dispose completely of the φ-items.

Notation 3.16 We denote one-step β-reduction using negative superscripts for φ by $t \to_{\beta'} t'$, and (ordinary) β-reduction — its reflexive and transitive closure — by $t \twoheadrightarrow_{\beta'} t'$. We write $=_{\beta'}$ for the equivalence relation generated by $\twoheadrightarrow_{\beta'}$. Again we use $\equiv$ for syntactic identity. Note that there are no β'-items.

Example 3.17
$(1\delta)(2\lambda)(4\delta)1 \to_{\beta'} (3\delta)1$ as follows:

$$\begin{array}{lll}
(1\delta)(2\lambda)(4\delta)1 & \to_{\sigma} & (1\delta)(2\lambda)((\varphi)1\sigma^{(1)})(4\delta)1 \\
& \to_{\sigma\varphi} & (1\delta)(2\lambda)((2\sigma^{(1)})4\delta)(2\sigma^{(1)})1 \\
& \twoheadrightarrow_{\sigma} & (1\delta)(2\lambda)(4\delta)2 \\
& \to_{\emptyset} & (\varphi^{(0,-1)})(4\delta)2 \\
& \to_{\varphi} & ((\varphi^{(0,-1)})4\delta)(\varphi^{(0,-1)})2 \\
& \twoheadrightarrow_{\varphi} & (3\delta)1.
\end{array}$$

Example 3.18
$(1\delta)(2\lambda)(3\lambda)2 \to_{\beta'} (2\lambda)2$ as follows:

$$\begin{array}{lll}
(1\delta)(2\lambda)(3\lambda)2 & \twoheadrightarrow_{\sigma\varphi} & (1\delta)(2\lambda)(3\lambda)3 \qquad \text{(see Example 3.11, 4)} \\
& \to_{\emptyset} & (\varphi^{(0,-1)})(3\lambda)3 \\
& \to_{\varphi} & ((\varphi^{(0,-1)})3\lambda)(\varphi^{(1,-1)})3 \\
& \twoheadrightarrow_{\varphi} & (2\lambda)2.
\end{array}$$

We shall not however in this paper use negative superscripts for φ in order to make a clear distinction between the harmless *positive* updating and the potentially dangerous *negative* updating (see our remark after Definition 3.14). Rather, we shall introduce a new kind of items $(\mu^{(i)})$ for $i \in I\!P$ with the same effect as $(\varphi^{(i,-1)})$ for void reductions. To be precise: $(\mu^{(i)})$ is equivalent to $(\varphi^{(i-1,-1)})$; but in the case of void reductions, $(\varphi^{(i-1,-1)})$ has the same effect as $(\varphi^{(i,-1)})$, as the reader may easily see.

3.3.2 β-reduction using $(\mu^{(i)})$

First we replace the void segment by $(\mu^{(1)})$. Then we let the $(\mu^{(i)})$ scan the term to its right doing the following:

- If $(\mu^{(i)})$ scans a λ then i increases by 1.
- If $(\mu^{(i)})$ scans a δ then nothing happens.
- If $(\mu^{(i)})$ reaches a superscript m then if $m \leq i$ nothing happens and if $m > i$ then m is decreased by 1.

Now the meaning of $(\mu^{(i)})t$ is: decrease all variables in t that are greater than i by an amount of 1. Those variables that are smaller or equal to i in t are bound by some λs in t and hence should not be decreased. Now the μ rules are defined as follows (recall that $(\mu^{(i)})$ occurs only in an $\Omega_{\lambda\delta}$-term):

Definition 3.19 *(μ-reduction)*
For all $t_1, t_2, t\ \Omega_{\lambda\delta}$-terms, $v \in \Xi$ and $i \in I\!P$ we have:

(μ-generation rule:)

$$(t_1\delta)(t_2\lambda)t \to_{\mu} (\mu^{(1)})t \quad \textit{if } (t_1\delta)(t_2\lambda) \textit{ is void in } t$$

(μ-transition rules:)

$$\begin{array}{lll}
(\mu^{(i)})(t\lambda) & \to_{\mu} & ((\mu^{(i)})t\lambda)(\mu^{(i+1)}) \\
(\mu^{(i)})(t\delta) & \to_{\mu} & ((\mu^{(i)})t\delta)(\mu^{(i)})
\end{array}$$

(μ-destruction rules:)

$$\begin{array}{lll} (\mu^{(i)})v & \to_\mu & v \qquad \text{if } v \equiv \varepsilon \text{ or } v < i \\ (\mu^{(i)})v & \to_\mu & v-1 \quad \text{if } i < v \end{array}$$

Note in the second μ-destruction rule that $v > 1$ as $i \geq 1$. Note moreover that we never reach the case where we get $(\mu^{(i)})i$ (see Lemma 3.22).

Similarly to σ- and φ-reduction, we implicitly assume the compatibility rules (see Remark 3.5) and we denote the reflexive and transitive closure of $\to_\mu$ by $\twoheadrightarrow_\mu$.

The one-step β-reduction that we assume in this paper hence will be based on this $(\mu^{(i)})$ and is defined as follows:

Definition 3.20 *(One-step β-reduction $\to_{\beta''}$)*
One-step β-reduction of an $\Omega_{\lambda\delta}$-term is the combination of one σ-generation from a $\delta\lambda$-segment $\overline{s}$, the transition of the generated σ-item through the appropriate subterm in a global manner, followed by a number of σ-destructions, and updated by φ-items until again an $\Omega_{\lambda\delta}$-term is obtained. Finally, we replace the now void segment $\overline{s}$ by $(\mu^{(1)})t$ and we use the μ-reduction rules to dispose completely of μ in $(\mu^{(1)})t$.

Finally we use the same notation as in Notation 3.16 except that we change β' to β''.

Example 3.21 $(4\delta)(\lambda)(1\lambda)(1\lambda)3 \to_{\beta''} (4\lambda)(1\lambda)6$:

$$\begin{array}{lll} (4\delta)(\lambda)(1\lambda)(1\lambda)3 & \to_\sigma & (4\delta)(\lambda)((\varphi)4\sigma^{(1)})(1\lambda)(1\lambda)3 \\ & \twoheadrightarrow_{\sigma,\varphi} & (4\delta)(\lambda)(5\lambda)(1\lambda)7 \\ & \to_\mu & (\mu^{(1)})(5\lambda)(1\lambda)7 \\ & \to_\mu & ((\mu^{(1)})5\lambda)(\mu^{(2)})(1\lambda)7 \\ & \to_\mu & (4\lambda)(\mu^{(2)})(1\lambda)7 \\ & \twoheadrightarrow_\mu & (4\lambda)(1\lambda)(\mu^{(3)})7 \\ & \to_\mu & (4\lambda)(1\lambda)6 \end{array}$$

The following lemma is needed when discussing the semantics of μ-reduction:

Lemma 3.22 *If t is an $\Omega_{\lambda\delta}$-term and $t \to_\mu t'$ then for all $(\mu^{(i)})t''$ subterm of t' with t'' an $\Omega_{\lambda\delta}$-term, we have that i does not refer to any free variable of t''. In particular, if $t \twoheadrightarrow_\mu t'$ then we never find in t', $(\mu^{(i)})i$ as a subterm.*

Proof: *By induction on $\twoheadrightarrow_\mu$.* □

4 Translating Λ in Ω_Ξ

Recall that we assume a free variable list $\mathcal{F}$, which is drawn in Figure 2. Let us enumerate this list in the order in which the variables appear from right to left. We call this enumeration function $\dagger$, so that:

$\dagger x_1 = 1, \dagger x_2 = 2, \dagger x_3 = 3, \ldots$.

We define moreover, for $v \in \mathcal{F}$, $\dagger\lambda_v$ to be λ, $\dagger\delta$ to be δ and $\dagger\varepsilon$ to be ε.

Now, let us take each term of Λ into a term of Ω_Ξ. For this we define the following notions:

Definition 4.1 *($term_i$)*
We define $term_i$ to be a partial function which takes non empty segments of Λ and returns terms of Λ as follows:

$term_1((t_1\omega_1)\overline{s}) =_{df} t_1$,

$term_i((t_1\omega_1)\overline{s}) =_{df} term_{i-1}(\overline{s})$, *for* $i \geq 2, \overline{s} \neq \emptyset$

Definition 4.2 *(lam_i)*
We define lam_i to be a function which takes a segment $\overline{s}$ of Λ and returns the segment $(\lambda_{v_1})(\lambda_{v_2})\ldots(\lambda_{v_k})$ obtained by removing all the main δ-items from the first $(i-1)$ main-items of $\overline{s}$ and by removing all the t's from the main λ-items $(t\lambda_v)$ of these $(i-1)$ main-items. lam_i is defined as follows:

$$\begin{array}{lll} lam_i(\overline{s}) & =_{df} & \emptyset \\ lam_i((t\lambda_v)\overline{s}) & =_{df} & (\lambda_v)lam_{i-1}(\overline{s}) \quad \text{for } i \geq 2 \text{ and } weight(\overline{s}) \geq i-2 \\ lam((t\delta)\overline{s}) & =_{df} & lam(\overline{s}) \quad\quad\quad\quad\quad \text{for } i \geq 2 \text{ and } weight(\overline{s}) \geq i-2 \end{array}$$

We take $Seq_{i=1}^{i=n}(t_i\omega_i)$ to stand for: $(t_1\omega_1)(t_2\omega_2)\ldots(t_n\omega_n)$, $n \geq 0$.

Now we define the translation as follows:

Definition 4.3 *(b)*
For $t, t_1, t_2 \in \Lambda, v, v' \in \mathcal{F}, \overline{s}$ segment of Λ, we define b, the translation function from Λ into Ω_Ξ as follows:

$$\begin{array}{lll} b(t) & =_{df} & b'(t, \emptyset) \\ b(\overline{s}) & =_{df} & \mathtt{body}(b(\overline{s}\varepsilon)) \\ b'(\varepsilon, \overline{s}) & =_{df} & \varepsilon \\ b'(v, \emptyset) & =_{df} & \dagger v \text{ (note } v \not\equiv \varepsilon) \\ b'(v, \overline{s}(\lambda_v)) & =_{df} & 1 \\ b'(v, \overline{s}(\lambda_{v'})) & =_{df} & 1 + b'(v, \overline{s}) \text{ if } v' \not\equiv v \\ b'((t_1\lambda_v)t_2, \overline{s}) & =_{df} & (b'(t_1, \overline{s})\lambda)b'(t_2, \overline{s}(\lambda_v)) \\ b'((t_1\delta)t_2, \overline{s}) & =_{df} & (b'(t_1, \overline{s})\delta)b'(t_2, \overline{s}) \end{array}$$

Here $b'(v, \overline{s})$ finds the de Bruijn number corresponding to v whithin context $\overline{s}$ (see Example 4.5). $b'((t_1\lambda_v)t_2, \overline{s})$ finds the translation of t_1 with respect to $\overline{s}$ and the translation of t_2 with respect to $\overline{s}(\lambda_v)$. $b'((t_1\delta)t_2, \overline{s})$ is now obvious.

Lemma 4.4
If $\overline{s_1}, \overline{s_2}$ are segments of $\Lambda, v \in \mathcal{F} \cup \{\varepsilon\}$, then
$b'(\overline{s_1}v, \overline{s_2}) = Seq_{i=1}^{i=n}(b'(term_i(\overline{s_1}), \overline{s_2}lam_i(\overline{s_1}))\dagger op_i(\overline{s_1}))b'(v, \overline{s_2}lam_{n+1}(\overline{s_1}))$, *for* $n = weight(\overline{s_1})$.

Proof: *By induction on the length of $\overline{s_1}$.* □

Essentially, what this lemma is saying is that, given a term t of the form $(t_1\omega_1)(t_2\omega_2)\ldots(t_n\omega_n)v \equiv \overline{s_1}v$ of Λ, then $b'(t, \overline{s_2}) = (t_1' \dagger \omega_1)(t_2' \dagger \omega_2)\ldots(t_n' \dagger \omega_n)v'$ where $t_i' \equiv b'(t_i, \overline{s_2}lam_i(\overline{s_1}))$ and $v' \equiv b'(v, \overline{s_2}lam_{n+1}(body(t)))$.

Hence, t and $b'(t, \overline{s_2})$ will have the same trees, except that all λ's lose their subscripts and all variables are replaced by the correct indices. These correct indices are found by tracing the λ's. That is why, in t_i', we had to attach all the λs preceding t_i'.

Now the following example illustrates how some terms of Λ can be translated in Ω_Ξ.

Example 4.5

1. $b((x_1\lambda_{x_4})(x_2\lambda_{x_3})x_4) \equiv (b'(x_1,\emptyset)\lambda)(b'(x_2,(\lambda_{x_4}))\lambda)b'(x_4,(\lambda_{x_4})(\lambda_{x_3})) \equiv (\dagger x_1\lambda)(3\lambda)2 \equiv (1\lambda)(3\lambda)2.$
2. $b((x_1\delta)(x_2\lambda_{x_4})(x_3\delta)x_4) \equiv (1\delta)(2\lambda)(4\delta)1.$
3. $b(((x_3\lambda_{x_4})x_4\delta)x_1) \equiv b'((x_3\lambda_{x_4})x_4,\emptyset)\delta)b'(x_1,\emptyset) \equiv ((b'(x_3,\emptyset)\lambda)b'(x_4,(\lambda_{x_4}))\delta)1 \equiv ((3\lambda)1\delta)1$

Lemma 4.6 *For any t in Λ, $b(t)$ is well defined.*
Proof: *By induction on $t \in \Lambda$.* □

Note that the translation function b is not injective. This is because $b((x_1\lambda_{x_2})x_2) \equiv b((x_1\lambda_{x_3})x_3)$ but $(x_1\lambda_{x_2})x_2 \not\equiv (x_1\lambda_{x_3})x_3$. b however is surjective but we will see this in Section 5 (see Lemma 5.9). For now the following lemma is informative about b.

Lemma 4.7 *If t, t' are terms in Λ such that $t =_\alpha t'$ then $b(t) \equiv b(t')$.*
Proof: *By induction on $t =_\alpha t'$.* □

5 Translating Ω_Ξ in Λ

Our first step in providing a semantics of substitution is to provide a translation of Ω_Ξ to Λ. In carrying out the translation we have to associate to each de Bruijn index a variable, which will be either free or bound in the term. We need to make sure of course that if a variable is free, then it will not become unintentionally bound by our choice of the name of a binding variable.

Example 5.1 In interpreting $(\lambda)2$, we may choose any of $(\lambda_{x_i})x_1$ for $i \neq 1$ to be the corresponding Λ-term. We cannot however take $(\lambda_{x_1})x_1$.

So as an x for the λ, we must choose x_i for $i \in I\!P$, but we must make sure that no free variable will have the same name as the chosen x_i. There is another case where we have to be careful. This is given in the following example:

Example 5.2 In interpreting the Ω_Ξ-subterms as Λ-terms, one should extend the free variable list in an obvious manner. For example, the term $t \equiv ((1\delta)2\lambda)(1\lambda)3$ has for any $i, j \neq 1$, $((x_1\delta)x_2\lambda_{x_i})(x_i\lambda_{x_j})x_1$ as a corresponding Λ-term. Now the subterm $(1\lambda)3$ of t should be considered relative to a free variable list extended with λ_{x_i}: $\ldots, \lambda_{x_4}, \lambda_{x_3}, \lambda_{x_2}, \lambda_{x_1}, \lambda_{x_i}$, and hence corresponds with $(x_i\lambda_{x_j})x_1$ for $j \neq 1$.

Now all this need to check whether the variable we choose now as the name of a bound variable will actually occur free in the term at some stage, pushes us to choose a less clumsy approach. The idea is to start from the list $\mathcal{F}$ which is given in Figure 2 and to work at a level between Ω_Ξ and Λ. In this *mid-level* $\overline{\Lambda}$, we always take the subscripts of λ's to be in a list $\updownarrow = x', x'', \ldots$ which is disjoint with $\mathcal{F}$. Now, there will be no danger that we might choose subscripts of λ's to be any x_i which will eventually occur in the term, as $\mathcal{F} \cap \updownarrow = \emptyset$.

Definition 5.3 *($\overline{\Lambda}$) The terms of $\overline{\Lambda}$ are defined similarly to those of Λ except that all bound variables are indexed by elements from $\updownarrow$ instead of elements from $\mathcal{F}$ as in Λ. Terms of $\overline{\Lambda}$ are written in the item notation, similarly to the terms of Ω_Ξ and Λ.*

Examples of terms of $\overline{\Lambda}$ are $\varepsilon, (x_1\lambda_{x'})x'$ and $(x_1\lambda_{x'})(x'\delta)x''$.

The notions of bound and free variables, substitution, α- and β-conversion or reduction, and $\equiv$ defined for Λ can be easily extended to $\overline{\Lambda}$. For example here's how substitution is extended.

Definition 5.4 *(Substitution in $\overline{\Lambda}$) If t, t' are terms in $\overline{\Lambda}$ (i.e. all bound variables are in $\updownarrow$, and all free variables are in $\mathcal{F} \cup \updownarrow$), and if $v \in \mathcal{F} \cup \updownarrow$, then $t[v := t']'$ is exactly defined as in Definition 2.13 except that, $[v := t']$ is replaced everywhere by $[v := t']'$, $[v' := v'']$ is replaced by $[v' := v'']'$ and in the last clause, $\mathcal{F}$ is replaced by $\updownarrow$.*

Notation 5.5 Similarly to Λ, we use $FV(t)$ and $BV(t)$ to find the free and bound variables of t in $\overline{\Lambda}$, even though this is an extension of FV and BV in Λ. We use $\overline{\alpha}$, $\overline{\beta}$ for the extended α and β-conversion/reduction, and as we saw above, we use $t[\theta := t']'$ for substitution in $\overline{\Lambda}$.

When all de Bruijn's indices in an $\Omega_{\lambda\delta}$-term t have been replaced by names from $\mathcal{F}$ and $\updownarrow$ obtaining a term t' in $\overline{\Lambda}$, we can easily map the term t' to Λ by replacing all the variables in $\updownarrow$ by variables in $\mathcal{F}$ which do not occur in the term. Now in order to assure the uniqueness of the translation (between $\Omega_\Xi, \overline{\Lambda}$ and Λ), and in order to avoid binding free variables, we take the following conventions:

1. We assume that $\updownarrow$ is ordered and that the order is $x', x'', \ldots$.
2. We assume that any two elements of $\updownarrow$ are distinct exactly as all variables in $\mathcal{F}$ are distinct.
3. We always take the first fresh variable X_i in $\updownarrow$ as a subscript to the λ in hand.

Now, we define the translation from a subclass of $\overline{\Lambda}$ to Λ as follows:

Definition 5.6 *(Translating $\overline{\Lambda}$ in Λ via τ) If t is a term in $\overline{\Lambda}$ such that $FV(t) \subseteq \mathcal{F}$ and $BV(t) \subseteq \updownarrow$ then we translate t to t' by first looking for the biggest free variable in t (recall $\mathcal{F}$ is ordered). Say this free variable is x_i for $i \in I\!P$. Now we take the smallest bound variable in t (recall $\updownarrow$ is ordered). We replace all the occurrences of this bound variable by x_{i+1}. Then we replace the second smallest bound variable by x_{i+2} and so on until no variables from $\updownarrow$ appear in t. We call the translation of the $\overline{\Lambda}$-term t in LT, $\tau(t)$.*

Note that this definition only translates t if $FV(t) \subseteq \mathcal{F}$ and $BV(t) \subseteq \updownarrow$. But not every term of $\overline{\Lambda}$ satisfies this property. All terms of $\overline{\Lambda}$ however which are translations of terms in $\Omega_\Xi^{\lambda\delta\sigma\varphi\mu}$ satisfy this property (see Lemma 5.53).

Example 5.7 The translation of $(\lambda)2$ in the mid-level $\overline{\Lambda}$ is $(\lambda_{x'})x_1$
The translation of $((1\delta)2\lambda)(1\lambda)3$ in the mid-level $\overline{\Lambda}$ is $((x_1\delta)x_2\lambda_{x'})(x'\lambda_{x''})x_1$.
Finally these terms in the mid-level are transformed into terms of Λ in a *unique* way as follows:
The greatest variable of $\mathcal{F}$ in $(\lambda_{x'})x_1$ is x_1, hence x' gets replaced by x_2, giving $(\lambda_{x_2})x_1$.
The greatest variable of $\mathcal{F}$ in $((x_1\delta)x_2\lambda_{x'})(x'\lambda_{x''})x_1$ is x_2, hence all occurrences of x', x'' get replaced by x_3, x_4 respectively giving $((x_1\delta)x_2\lambda_{x_3})(x_3\lambda_{x_4})x_1$.

Now, as Λ and $\overline{\Lambda}$ are very similar, we shall avoid the trivial step of translating between Λ and $\overline{\Lambda}$ and shall show the soundness in $\overline{\Lambda}$. The reader can see however that this simplification does not affect any of the results of this paper.

But, how do we provide this translation which takes $\Omega_{\lambda\delta\sigma\varphi\mu}$-terms to the mid-level? This we may start as follows:

5.1 The inverse function e

We may give the definition of the function e which takes elements of $\Omega^{\lambda\delta}_{\underline{\Xi}}$ to the mid-level mentioned above as follows:

Definition 5.8 *(e)*
Let $t, t_1, t_2 \in \Omega^{\lambda\delta}_{\Xi}, \overline{s}$ be a segment of $\overline{\Lambda}$ consisting of items of the form (λ_X) for $X \in \updownarrow, l \in \mathcal{L}_\infty(\updownarrow), j \in I\!P, v \in \Xi, X \in \updownarrow$. The function e which takes $\Omega_{\lambda\delta}$-terms into terms in $\overline{\Lambda}$ (which use variables in $\mathcal{F} \cup \updownarrow$) is defined as follows:

$$
\begin{array}{lll}
e(t) & =_{df} & c(t, \emptyset, \updownarrow) \\
c(v, \overline{s}, l) & =_{df} & d(v, \overline{s}) \\
c((t_1\delta)t_2, \overline{s}, l) & =_{df} & (c(t_1, \overline{s}, l)\delta)c(t_2, \overline{s}, tl^{nl(t_1)}(l)) \\
c((t_1\lambda)t_2, \overline{s}, l) & =_{df} & (c(t_1, \overline{s}, l)\lambda_{hd^{1+nl(t_1)}(l)})c(t_2, \overline{s}(\lambda_{hd^{1+nl(t_1)}(l)}), tl^{1+nl(t_1)}(l)) \\
d(j, \emptyset) & =_{df} & x_j \\
d(\varepsilon, \overline{s}) & =_{df} & \varepsilon \\
d(1, \overline{s}(\lambda_X)) & =_{df} & X \\
d(n, \overline{s}(\lambda_X)) & =_{df} & d(n-1, \overline{s}) \text{ if } n > 1
\end{array}
$$

Here $\mathcal{L}_\infty(\updownarrow)$ is the set of those sublists of $\updownarrow$ which are equal to $\updownarrow$ with an initial segment removed (see Definition 5.16). Moreover, we take hd^i and tl^i, for $i \geq 1$, to be functions which take lists and return the i^{th} element of the list, respectively the list without its first i elements (see Section 5.2). Recall moreover that $nl(t)$ is defined to be the number of λ's in t (see Definition 2.25).

Note that d associates with each de Bruijn's index, the right variable in $\mathcal{F} \cup \updownarrow$ which should replace it.

Lemma 5.9 *e is well defined and $b \circ \tau \circ e(t) \equiv t$ for any $t \in \Omega^{\lambda\delta}_{\underline{\Xi}}$*
Proof: *Easy.* □

Example 5.10

$$
\begin{array}{lll}
e((2\lambda)2\lambda)1 & \equiv & c(((2\lambda)2\lambda)1, \emptyset, \updownarrow) \\
& \equiv & (c((2\lambda)2, \emptyset, \updownarrow)\lambda_{x''})c(1, (\lambda_{x''}), \{x''', x^{iv}, \ldots\}) \\
& \equiv & ((c(2, \emptyset, \updownarrow)\lambda_{x'})c(2, (\lambda_{x'}), \{x'', x''', \ldots\})\lambda_{x''})d(1, (\lambda_{x''})) \\
& \equiv & ((d(2, \emptyset)\lambda_{x'})d(2, (\lambda_{x'}))\lambda_{x''})x'' \\
& \equiv & ((x_2\lambda_{x'})d(1, \emptyset)\lambda_{x''})x'' \\
& \equiv & ((x_2\lambda_{x'})x_1\lambda_{x''})x''
\end{array}
$$

(Note that the first λ to be be named becomes $\lambda_{x''}$ and not $\lambda_{x'}$, due to the fact that there is one λ in $(2\lambda)2$; i.e. $nl((2\lambda)2) = 1$, hence $\lambda_{hd^{1+nl((2\lambda)2)}(\updownarrow)} = \lambda_{hd^2(\updownarrow)} = \lambda_{x''}$.) This $\overline{\Lambda}$-term may be replaced by the term $((x_2\lambda_{x_3})x_1\lambda_{x_4})x_4$ in Λ.

Example 5.11

$$
\begin{aligned}
e((\lambda)(1\lambda)(1\delta)3) &\equiv c((\lambda)(1\lambda)(1\delta)3, \emptyset, \updownarrow) \\
&\equiv (c(\varepsilon, \emptyset, \updownarrow)\lambda_{x'})c((1\lambda)(1\delta)3, (\lambda_{x'}), \{x'', x''', \ldots\}) \\
&\equiv (d(\varepsilon, \emptyset)\lambda_{x'})(c(1, (\lambda_{x'}), \{x'', x''', \ldots\})\lambda_{x''})c((1\delta)3, (\lambda_{x'})(\lambda_{x''}), \{x''', \ldots\}) \\
&\equiv (\varepsilon\lambda_{x'})(d(1, (\lambda_{x'}))\lambda_{x''})(c(1, (\lambda_{x'})(\lambda_{x''}), \{x''', \ldots\})\delta)c(3, (\lambda_{x'})(\lambda_{x''}), \{x''', \ldots\}) \\
&\equiv (\varepsilon\lambda_{x'})(x'\lambda_{x''})(d(1, (\lambda_{x'})(\lambda_{x''}))\delta)d(3, (\lambda_{x'})(\lambda_{x''})) \\
&\equiv (\lambda_{x'})(x'\lambda_{x''})(x''\delta)d(2, (\lambda_{x'})) \\
&\equiv (\lambda_{x'})(x'\lambda_{x''})(x''\delta)d(1, \emptyset) \\
&\equiv (\lambda_{x'})(x'\lambda_{x''})(x''\delta)x_1
\end{aligned}
$$

Finally, we get rid of the variables of $\updownarrow$ in $(\lambda_{x'})(x'\lambda_{x''})(x''\delta)x_1$ by replacing every x' by x_2 and every x'' by x_3 obtaining $(\lambda_{x_2})(x_2\lambda_{x_3})(x_3\delta)x_1$

This e however does not take into account φ-, σ- and μ-items. In fact, it is difficult to provide the translation of φ-items without watching what happens in the lists $\mathcal{F}$ and $\updownarrow$. Look at the following example:

Example 5.12 Take the term in Ω_Ξ to be $(\varphi^{(1,2)})(1\delta)(2\lambda)3$. Now, the translation of this term should be: $(x_1\delta)(x_4\lambda_{x'})x_4$ and will finally be transformed into the Λ-term $(x_1\delta)(x_4\lambda_{x_5})x_4$. What this really mean is that due to the presence of $(\varphi^{(1,2)})$, we translate $(1\delta)(2\lambda)3$ not in terms of $\mathcal{F}$ and $\updownarrow$ as we have done so far, but in terms of $\mathcal{F}'$ and $\updownarrow$ where $\mathcal{F}' = \ldots x_5{+}{+}x_4{+}{+}x_1$. I.e. the x_2 and x_3 disappear from $\mathcal{F}$. (For lists notation, see the following section.)

This process of removing elements from $\mathcal{F}$ must also be extended to sublists of $\mathcal{F} \cup \updownarrow$ in order to translate subterms of terms. Moreover, we need, in order to show the correctness of our tranlation and the soundness of our reduction rules, to have some basic formulation of lists. We start therefore by setting the ground for these lists.

5.2 Variables and lists

Definition 5.13 *(Θ)*
We define the set of variables Θ to be $\updownarrow \cup \mathcal{F}$. We let $\theta, \theta_1, \theta_2, \theta', \cdots$ range over Θ. Note that $\varepsilon \notin \Theta$. Recall moreover that $v, v', v_1, v_2, \ldots$ range over $\mathcal{F}$, that $\mathcal{F}$ has $x_1, x_2, \ldots$ for elements and that $\updownarrow = x', x'', \ldots$. Furthermore, we take $X, X', X_1, X_2, \ldots$ to range over $\updownarrow$. We refer sometimes to elements of $\mathcal{F}$ as free variables and to elements of $\updownarrow$ as bound variables.

Now, we will use lists as an important part of our semantic function. We assume the usual basic list operations such as concatenation $+{+}$ and *head* and *tail*, *hd* and *tl*. For $i \in \mathbb{P}$, we take $hd^1 =_{df} hd$ and $hd^{i+1} =_{df} hd \circ hd^i$, and we define tl^i similarly. Moreover, the set of operators $\setminus, \subset, \subseteq$ and $\in$ are also applicable for lists and we will mix sets and lists at will. We take $\overline{v}, \overline{v'}, \overline{v_1}, \overline{v_2}, \ldots$ to range over (finite and infinite) lists.

As we have seen in Example 5.12, we need to add/remove variables from $\mathcal{F}$ due to the updating function $(\varphi^{k,i})$. Hence we define the following notions related to lists:

Definition 5.14 *(reversed list of variables, left part, right part)*

- *Every list is written as the sum of its ordered elements from right to left In particular, we write* $\mathcal{F}$ *as* $\ldots + +x_2 + +x_1$ *and* $\updownarrow$ *as* $\ldots + +x'' + +x'$.
- *If* $\overline{v} = \ldots + +\theta_2 + +\theta_1$, *then for* $m \geq 1$, *we define* $\overline{v}_{\geq m}$ *to be* $\ldots + +\theta_{m+1} + +\theta_m$. $\overline{v}_{\geq m}$ *is also called the left part of* $\overline{v}$ *starting at* m. *Note that* $\overline{v}_{\geq m} = tl^{m-1}(\overline{v})$. *In particular, we define* $\mathcal{F}_{\geq m}$ *to be* $\ldots x_{m+1} + +x_m$ *for* $m \geq 1$.
- *If* $\overline{v} = \ldots + +\theta_2 + +\theta_1$, *then for* $m \geq 1$, *we define* $\overline{v}_{<m}$ *to be* $\theta_{m-1} + +\theta_{m-2} + + \ldots + +\theta_1$. *Note that* $\overline{v}_{<1}$ *is the empty list and* $\overline{v}_{<2} = hd(\overline{v})$. $\overline{v}_{<m}$ *is also called the right part of* $\overline{v}$ *ending before* m. *In particular, we define* $\mathcal{F}_{<m}$ *to be* $x_{m-1} + +x_{m-2} + + \ldots + +x_1$ *for* $m \geq 1$.

Definition 5.15 *($\mathcal{L}$) If $\mathcal{A}$ is a set, then we define $\mathcal{L}(\mathcal{A})$ to be the set of all finite lists generated by $\mathcal{A}$. We assume that all elements $a \in \mathcal{A}$ occur at most once, in each of these finite lists. Obviously, the empty list $\emptyset \in \mathcal{L}(\mathcal{A})$ for every set $\mathcal{A}$.*

Note that $\mathcal{L}$ only generates finite lists. In particular, $\updownarrow \notin \mathcal{L}(\updownarrow)$.

Definition 5.16 *($\mathcal{L}_\infty(\overline{v})$)*
We define $\mathcal{L}_\infty(\overline{v})$ to be $\{\overline{v}_{\geq i}; i \in I\!P\}$. I.e. elements of $\mathcal{L}_\infty(\overline{v})$ are $\overline{v}, tl(\overline{v})$ and so on.

Lists that we will be using often are those for whom a right part is a finite list of elements of $\Theta \cup \{\psi\}$ (where ψ is a special symbol $\notin \Theta$ whose meaning for lists will become clear below), and a left part is $\mathcal{F}_{\geq m}$ for some $m \in I\!P$. For this reason, we define the following:

Definition 5.17 *($\mathcal{L}_{split}$)*
$\mathcal{L}_{split}$ is defined to be: $\{\mathcal{F}_{\geq m} + +\overline{v}; m \in I\!P, \overline{v} \in \mathcal{L}(\Theta \cup \{\psi\})\}$

Hence, if $\overline{v} \in \mathcal{L}_{split}$ then $\overline{v}$ can be split up in two lists: $\overline{v} \equiv \mathcal{F}_{\geq m} + +\overline{v'}$.

1. The left sublist, is an infinite left part of $\mathcal{F}$.
2. The right sublist is an element of $\mathcal{L}(\Theta \cup \{\psi\})$. That is, a finite list of elements from $\Theta \cup \{\psi\}$.

Definition 5.18 *($\mathcal{L}^{-1}(\Theta)$)*
We define $\mathcal{L}^{-1}(\Theta)$ to be: $\{\overline{v}; \overline{v} \in \mathcal{L}_{split} \wedge \overline{v}$ is ψ-free$\}$. I.e. elements of $\mathcal{L}^{-1}(\Theta)$ are those elements of $\mathcal{L}_{split}$ which do not contain ψ.

Definition 5.19 *($\mathcal{L}_\psi$)*
We define $\mathcal{L}_\psi$ to be $\mathcal{L}_{split} \cup \mathcal{L}(\Theta \cup \{\psi\})$.

Now the following function intends to measure the length of finite lists in which ψ appears. From this function, the reader can guess that ψ removes an element from the set.

Definition 5.20 *The function $||\cdot|| : \mathcal{L}(\Theta \cup \{\psi\}) \longmapsto \mathbb{Z}$ is defined as follows:*
For all $\overline{v} \in \mathcal{L}(\Theta \cup \{\psi\}), \theta \in \Theta$:

$$\begin{array}{lcl} ||\emptyset|| & = & 0 \\ ||\overline{v} + +\psi|| & = & ||\overline{v}|| - 1 \\ ||\overline{v} + +\theta|| & = & ||\overline{v}|| + 1 \end{array}$$

We write $|\overline{v}|$ for the length of $\overline{v}$ (i.e. the number of all its elements including ψ).

Lemma 5.21 *For all* $\overline{v} \in \mathcal{L}(\Theta \cup \{\psi\})$, $||\overline{v}|| \leq |\overline{v}|$. *Moreover, if* $\overline{v} \in \mathcal{L}(\Theta)$ *then* $||\overline{v}|| = |\overline{v}|$.
Proof: *Obvious.* □

Moreover, we define the following *partial* function:

Definition 5.22 *(comp) For all* $\overline{v} \in \mathcal{L}_{\psi}, \theta \in \Theta, n \in I\!P$:

$$\begin{array}{lll} comp_1(\overline{v} + +\theta) & =_{df} & \theta \\ comp_{n+1}(\overline{v} + +\theta) & =_{df} & comp_n(\overline{v}) \\ comp_n(\overline{v} + +\theta + +\psi^{i+1}) & =_{df} & comp_n(\overline{v} + +\psi^i), \quad i \in I\!N \end{array}$$

Here ψ^n stands for $\underbrace{\psi + + \ldots + +\psi}_{n}$ for $n \in I\!N$.

The idea of *comp* is to select the appropriate named variable, given a list of (different) named variables. We write $comp_n(\overline{v})\uparrow$, when $comp_n(\overline{v})$ is defined.

Lemma 5.23 *For all* $\overline{v} \in \mathcal{L}(\Theta \cup \{\psi\}), n \in I\!P$, *if* $n \leq ||\overline{v}||$ *then* $comp_n(\overline{v})\uparrow \wedge comp_n(\overline{v}) \in \overline{v}$.
Proof: *By induction on* $|\overline{v}|$ *noting that if* $||\overline{v}|| \geq 1$ *then* $\exists \theta \in \Theta$ *such that* $\theta \in \overline{v}$. □

Corollary 5.24 *For all* $\overline{v} \in \mathcal{L}(\Theta), n \in I\!P$, *if* $n \leq |\overline{v}|$ *then* $comp_n(\overline{v})\uparrow \wedge comp_n(\overline{v}) \in \overline{v}$.
Proof: *Obvious, using Lemmas 5.21 and 5.23.* □

Lemma 5.25 *For all* $\overline{v} \in \mathcal{L}_{split}, n \in I\!P, comp_n(\overline{v})\uparrow \wedge comp_n(\overline{v}) \in \overline{v}$.
Proof: *By induction on* n. □

Note that the only case where $comp_n(\overline{v})$ is undefined is when $n > ||\overline{v}||$.

Lemma 5.26 *For all* $\overline{v} \in \mathcal{L}_{split}, n \in I\!P, i \in I\!N, comp_n(\overline{v} + +\psi^i) = comp_{n+i}(\overline{v})$.
Proof: *Easy.* □

Lemma 5.27 *For all* $\overline{v'} \in \mathcal{L}_{split}, \overline{v} \in \mathcal{L}(\Theta \cup \{\psi\}), \theta \in \Theta, n \in I\!P$, *and* $i \in I\!N$, *we have:*

1. *If* $n > ||\overline{v}|| \geq 0$ *then* $comp_n(\overline{v'} + +\overline{v}) \equiv comp_{n-||\overline{v}||}(\overline{v'})$.
2. *If* $n > ||\overline{v}|| \geq 0$ *then* $comp_n(\overline{v'} + +\psi^i + +\overline{v}) \equiv comp_{n+i}(\overline{v'} + +\overline{v})$.
3. *If* $n \leq ||\overline{v}||$ *then* $comp_n(\overline{v'} + +\overline{v}) \equiv comp_n(\overline{v})$.
4. $comp_n(\overline{v'} + +\theta + +\psi + +\overline{v}) \equiv comp_n(\overline{v'} + +\overline{v})$.

Proof:

1. *By induction on* $|\overline{v}|$ *using Lemma 5.26.*
2. *Using Lemma 5.26 and 1 above.*

3. *By induction on* $|\overline{v}|$ *using Lemma 5.26.*

4. - *Case* $n \leq ||\overline{v}||$ *or* $n > ||\overline{v}|| \geq 0$, *then use the definition of comp and cases 1 and 3 above.*
   - *Case* $n > ||\overline{v}||$ *and* $||\overline{v}|| < 0$ *then by induction on* $|\overline{v}|$.

□

Finally, the following definition takes a segment to the list of variables which are indices of the λs occurring in the main items of the segment.

Definition 5.28
If $\overline{s}$ *is a segment, then we define the list based on* s *to be as follows:* $sl(\emptyset) = \emptyset$, $sl((t_1\delta)\overline{s'}) = sl(\overline{s'})$ *and* $sl((t\lambda_\theta)\overline{s'}) = \theta + +sl(\overline{s'})$.

5.3 The semantics of Ω_Ξ-terms: an initial account

The method here is to provide the semantics of the terms using lists of variables $\overline{v}$ and $\overline{v'}$ so that $\lfloor\overline{v};\overline{v'};t'\rfloor$ where t' is a subterm of t searches for the translation of $t' \in \Omega_\Xi$ using $\overline{v}$ to give names to the free variables in t' and $\overline{v'}$ to give names to the bound variables in t'. Moreover, $\overline{v} \cap \overline{v'}$ is taken to be $\emptyset$ in order to avoid binding any free variable.

Now, if we were to determine the semantics of the λ- and δ-terms only, then it is sufficient to consider $\overline{v} \in \mathcal{L}(\updownarrow)$ as we have done in the definition of e in Definition 5.8. The list $\overline{v}$ then may be considered as the list of named variables to be used for free variables in t' which are bound in the original term t; variables free in t obtain their names relative to the fixed list $\ldots x_3 + +x_2 + +x_1$. With variable updating however, we will consider $\overline{v}$ to be denumerably infinite and in $\mathcal{L}_{split}$. We start first with only finite lists of elements of $\updownarrow$ and we provide the semantics of the λ- and δ-terms as follows:

Definition 5.29 *(λ- and δ-semantics)*
For all $t_1, t_2 \in \Omega_\Xi^{\lambda\delta}, \overline{v} \in \mathcal{L}(\updownarrow), \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, n \in I\!P \cup \{\varepsilon\}$,

$$\begin{array}{lll}
\lfloor\overline{v};\overline{v'};(t_1\lambda)t_2\rfloor & =_{df} & (\lfloor\overline{v};\overline{v'};t_1\rfloor\lambda_X)\lfloor\overline{v} + +X;\overline{v'}_{\geq i+1};t_2\rfloor \textit{ for } i = nl(t_1)+1, X = hd^i(\overline{v'}) \\
\lfloor\overline{v};\overline{v'};(t_1\delta)t_2\rfloor & =_{df} & (\lfloor\overline{v};\overline{v'};t_1\rfloor\delta)\lfloor\overline{v};\overline{v'}_{\geq i};t_2\rfloor \textit{ for } i = nl(t_1)+1 \\
\lfloor\overline{v};\overline{v'};n\rfloor & =_{df} & \begin{cases} comp_n(\overline{v}) & \textit{if } n \leq |\overline{v}| \\ x_{n-|\overline{v}|} & n > |\overline{v}| \\ \varepsilon & \textit{if } n = \varepsilon \end{cases}
\end{array}$$

That is, we save in $\overline{v}$ all those variables which are now free in the term we are calculating, but which were bound originally. Note that the condition $\overline{v} \cap \overline{v'} = \emptyset$ is necessary; otherwise we would bind variables that are meant to be free.

Example 5.30 *(see Example 5.10)*

$$\begin{array}{ll}
\lfloor\emptyset;\updownarrow;((2\lambda)2\lambda)1\rfloor & \equiv \\
(\lfloor\emptyset;\updownarrow;(2\lambda)2\rfloor\lambda_{x''})\lfloor x'';\updownarrow_{\geq 3};1\rfloor & \equiv \\
((\lfloor\emptyset;\updownarrow;2\rfloor\lambda_{x'})\lfloor x';\updownarrow_{\geq 2};2\rfloor\lambda_{x''})comp_1(x'') & \equiv \\
((x_{2-|\emptyset|}\lambda_{x'})x_{2-|x'|}\lambda_{x''})x'' & \equiv \\
((x_2\lambda_{x'})x_1\lambda_{x''})x'' & \equiv
\end{array}$$

Example 5.31 *(see Example 5.11)*

$$
\begin{array}{ll}
\lfloor\emptyset;\updownarrow;(\lambda)(1\lambda)(1\delta)3\rfloor & \equiv \\
(\lfloor\emptyset;\updownarrow;\varepsilon\rfloor\lambda_{x'})\lfloor x';\updownarrow_{\geq 2};(1\lambda)(1\delta)3\rfloor & \equiv \\
(\varepsilon\lambda_{x'})(\lfloor x';\updownarrow_{\geq 2};1\rfloor\lambda_{x''})\lfloor x'x'';\updownarrow_{\geq 3};(1\delta)3\rfloor & \equiv \\
(\varepsilon\lambda_{x'})(comp_1(x')\lambda_{x''})(\lfloor x'x'';\updownarrow_{\geq 3};1\rfloor\delta)\lfloor x'x'';\updownarrow_{\geq 3};3\rfloor & \equiv \\
(\varepsilon\lambda_{x'})(x'\lambda_{x''})(comp_1(x'x'')\delta)x_{3-|x'x''|} & \equiv \\
(\varepsilon\lambda_{x'})(x'\lambda_{x''})(x''\delta)x_1 &
\end{array}
$$

If however we calculate $\lfloor x';\updownarrow;(\lambda)(1\lambda)(1\delta)3\rfloor$, then we would get $(\varepsilon\lambda_{x'})(x'\lambda_{x''})(x''\delta)x'$ which is not the intended meaning for $(\lambda)(1\lambda)(1\delta)3$. Note that the list v' is superfluous when we always start with $\lfloor\emptyset;\updownarrow;t'\rfloor$, since then $v' \equiv \updownarrow_{\geq|v|+1}$ and remains so.

Lemma 5.32 *For any* $\overline{v} \in \mathcal{L}(\updownarrow), \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, t \in \Omega_\Xi^{\lambda\delta}, FV(\lfloor\overline{v};\overline{v'};t\rfloor) \subseteq \overline{v} \cup \mathcal{F}$.
Proof: *By induction on* t*, recalling that* ε *is neither free nor bound.* □

Lemma 5.33 $\lfloor\cdot;\cdot;\cdot\rfloor$ *as defined in Definition 5.29 is well defined. That is for all* $\overline{v} \in \mathcal{L}(\updownarrow), \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, t \in \Omega_\Xi^{\lambda\delta}$, $\lfloor\overline{v};\overline{v'};t\rfloor$ *is a unique term of* $\overline{\Lambda}$.
Proof: *By induction on* $t \in \Omega_\Xi^{\lambda\delta}$ *using Corollary 5.24.* □

Now we will prove that e and $\lfloor\emptyset;\updownarrow;\cdot\rfloor$ return the same $\overline{\Lambda}$-terms.

Lemma 5.34 *For all* $t \in \Omega_\Xi^{\lambda\delta}, \overline{s}$ *segment from the mid-level and* $\overline{v} \in \mathcal{L}_\infty(\updownarrow), e(t, \overline{s}, \overline{v}) \equiv \lfloor sl(\overline{s});\overline{v};t\rfloor$.
Proof: *By induction on* t. □

Corollary 5.35 *For all* $t \in \Omega_\Xi^{\lambda\delta}, e(t) \equiv \lfloor\emptyset;\updownarrow;t\rfloor$.
Proof: *Obvious.* □

5.4 Extending the initial account

We have not so far, in either the translation using e or that of $\lfloor\cdot;\cdot;\cdot\rfloor$, defined the meaning of σ-items and φ-items. The meaning of the first is straightforward. In fact, for $i \in I\!P$, $t_1, t_2 \in \Omega_\Xi^{\lambda\delta}, \overline{v} \in \mathcal{L}(\updownarrow)$ and $\overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset$, we shall define:

$$\lfloor\overline{v};\overline{v'};(t_1\sigma^{(i)})t_2\rfloor =_{df} \lfloor\overline{v};\overline{v'};t_2\rfloor[\lfloor\overline{v};\overline{v'};i\rfloor := \lfloor\overline{v};\overline{v'}_{\geq 1+nl(t_2)};t_1\rfloor]'$$

where $t_1[v := t_2]'$ is the substitution in the mid-level $\overline{\Lambda}$ (which uses $\mathcal{F} \cup \updownarrow$) given in Definition 5.4.

When it comes to the meaning of $\lfloor\overline{v};\overline{v'};(\varphi^{(k,i)})t\rfloor$, then things may not be obvious. In fact, the intended meaning of $(\varphi^{(k,i)})t$ is: add i to all free variables greater than k, occurring within term t. Let us moreover summarize what our semantic function does. In $\lfloor\overline{v};\overline{v'};t\rfloor$, the term t is written exactly as it is (i.e. λ's and δ's stay at their original positions in t). The free

variables in t however (which are indices of course) are replaced by variables from $\overline{v} \cup \mathcal{F}$ (see Lemma 5.32). The index itself decides which variable from $\overline{v} \cup \mathcal{F}$ is to replace it. For example

$$\begin{array}{ll} \lfloor x'''x''x'; \updownarrow_{\geq 4}; (1\delta)(2\delta)3\rfloor & \equiv (x'\delta)(x''\delta)x''' \\ \lfloor x'''x''x'; \updownarrow_{\geq 4}; (1\delta)(2\lambda)1\rfloor & \equiv (x'\delta)(x''\lambda_{x'})x' \\ \lfloor x'; \updownarrow_{\geq 2}; 2\rfloor & \equiv x_1 \end{array}$$

Now, when we come to look for the meaning of $\lfloor \overline{v}; \overline{v'}; (\varphi^{(k,i)}t\rfloor$, then all those variables in t which are smaller than or equal to k, take the same value as if we were only calculating $\lfloor \overline{v}; \overline{v'}; t\rfloor$. Those variables bigger than k must not take the original values they would have taken in $\lfloor \overline{v}; \overline{v'}; t\rfloor$. Rather, looking for their corresponding variables in $\overline{v}$, we have to shift still i positions to the left. I.e. if the index is n, where $n > k$ then the variable corresponding to n is not the n^{th} variable from right to left in $\overline{v}$. Rather, it is the $(n+i)^{th}$ variable from the right. For example:

$$\lfloor x''''x'''x''x'; \updownarrow_{\geq 5}; (\varphi^{(1,2)})(1\delta)2\rfloor \equiv (x'\delta)x''''$$

Hence to calculate, let us say, $\lfloor \overline{v}; \overline{v'}; (\varphi^{(k,i)})t\rfloor$, we have to consider several cases:

- Case $|\overline{v}| \geq k + i$. Then the trailing k elements of list $\overline{v}$ are to be kept but the next i elements are to be erased resulting in a list $\overline{v_1} = left(\overline{v}, |\overline{v}| - k - i) + + right(\overline{v}, k)$ where $left$ and $right$ have the obvious meaning. I.e. $left(\overline{v}, m) = \overline{v}_{\geq m}$, $right(\overline{v}, m) = \overline{v}_{<m}$. Hence,

  $$\lfloor \overline{v}; \overline{v'}; (\varphi^{(k,i)})t\rfloor \equiv \lfloor \overline{v_1}; \overline{v'}; t\rfloor$$

  For example: $\lfloor x''''x'''x''x'; \updownarrow_{\geq 5}; (\varphi^{(1,2)})2\rfloor \equiv \lfloor x''''x'; \updownarrow_{\geq 5}; 2\rfloor \equiv x''''$.

- Case $|\overline{v}| < k$ where $\overline{v} \in \mathcal{L}(\updownarrow)$. Each free variable n in t, greater than k has to be increased by i. Now because $|\overline{v}| < k+i < n+i$, such a free variable will be associated with $x_{n-|\overline{v}|+i}$. For example, $\lfloor x'; \updownarrow_{\geq 2}; (\varphi^{(2,3)})3\rfloor \equiv x_5$ and $\lfloor \emptyset; \updownarrow; (\varphi^{(2,3)})3\rfloor \equiv x_6$. For a free variable n in t with $n \leq k$, nothing changes: take $x_{n-|\overline{v}|}$. For example: $\lfloor x'; \updownarrow_{\geq 2}; \varphi^{(2,3)}2\rfloor \equiv x_1$.

- Case $k \leq |\overline{v}| < k + i$. This is a mixture of the above two cases. For example

  $$\lfloor x''x'; \updownarrow_{\geq 3}; (\varphi^{(1,2)})(1\delta)2\rfloor = (x'\delta)x_2$$

In all these cases, the list $\overline{v}$ has to be updated, when calculating φ-items. There are essentially two ways to update the list so that the above three cases are accommodated. The first alternative will be called eager erasing and conceptually consists in immediately erasing the superfluous elements in $\overline{x}$. The second alternative is a stepwise approach and will be named *lazy erasing.*

Eager erasing just deletes the elements. So, if $|\overline{v}| \geq k + i$, then some function like $\lfloor \overline{v}; \overline{v'}; (\varphi^{(k,i)})t\rfloor \equiv \lfloor (left(\overline{v}, |\overline{v}| - k - i) + + right(\overline{v}, k)); \overline{v'}; t\rfloor$ would do the job.

Now for lazy erasing, the trick is to allow a special symbol ψ to become an element of $\overline{v}$. The operational meaning of ψ is: on going left, delete the first named variable. We will use lazy erasing in this paper. Moreover, as is traditional with our approach, we will use ψ with superscripts. We write ψ^1 as ψ and ψ^0 as the empty string $\emptyset$. ψ^n will be $\underbrace{\psi + + \cdots + +\psi}_{n}$.

Such a ψ, will not only be used to erase variables but will also say which free variable in $\mathcal{F}$ correponds to the variable in hand.

Example 5.36 The idea is that:

1. To calculate $\lfloor \overline{v}; \overline{v'}; (\varphi^{(k,i)})t \rfloor$ where $|\overline{v}| \geq k+i, \overline{v} = \overline{v_1} + +\overline{v_2}$ and $|\overline{v_2}| = k$, we calculate $\lfloor \overline{v_1} + +\psi^i + +\overline{v_2}; \overline{v'}; t \rfloor$. Hence when calculating $\lfloor x''''x'''x''x'; \updownarrow_{\geq 5}; (\varphi^{(1,2)})2 \rfloor$, we calculate $\lfloor x''''x'''x'' + +\psi^2 + +x'; \updownarrow_{\geq 5}; 2 \rfloor$. Now, this evaluates to $\lfloor x''''x'''x'' + +\psi^2; \updownarrow_{\geq 5}; 1 \rfloor$. The presence of ψ^2 means ignore $x'''x''$. Therefore the result reduces to $\lfloor x''''; \updownarrow_{\geq 5}; 1 \rfloor$ which is x''''.

2. For every $n \in I\!N, m \in I\!P, \lfloor \overline{v} + +\psi^n; \overline{v'}; m \rfloor = \lfloor \overline{v}; \overline{v'}; n+m \rfloor$ and $\lfloor \psi^n; \overline{v'}; m \rfloor = x_{n+m}$.

Looking at the first part of Example 5.36, we see that we need to have $\overline{v} = \overline{v_1} + +\overline{v_2}$ where $|\overline{v_2}| = k$. Now, we are interested in a stepwise fashion. Moreover, the length of $\overline{v_2}$ has to be calculated somehow. In other words, we have to go through the list $\overline{v}$ from right to left until we pass the k^{th} element. In order to accommodate such a stepwise fashion, we introduce an extra argument in the semantic meaning of φ-terms. We will give an example which explains the point even though it is ahead of its time in the section. We believe however, that the reader can still follow it, once point 2 of Example 5.36 is remembered.

Example 5.37 Notice how we save x' to use it later on:

$$
\begin{array}{ll}
\lfloor x''x'; \updownarrow_{\geq 3}; (\varphi^{(1,2)})(1\delta)2 \rfloor & \equiv \\
\lfloor x''; x'; \updownarrow_{\geq 3}; (\varphi^{(1,2)})(1\delta)2 \rfloor & \equiv \\
\lfloor x'' + +\psi^2 + +x'; \updownarrow_{\geq 3}; (1\delta)2 \rfloor & \equiv \\
(\lfloor x'' + +\psi^2 + +x'; \updownarrow_{\geq 3}; 1 \rfloor \delta) \lfloor x'' + +\psi^2 + +x'; \updownarrow_{\geq 3}; 2 \rfloor & \equiv \\
(x'\delta) \lfloor x'' + +\psi^2; \updownarrow_{\geq 3}; 1 \rfloor & \equiv \\
(x'\delta) \lfloor x''; \updownarrow_{\geq 3}; 3 \rfloor & \equiv \\
(x'\delta) x_2 &
\end{array}
$$

For reasons that will become clear below, we extend our lists from being elements of $\mathcal{L}(\updownarrow)$ (as in Definition 5.29) to being elements of $\mathcal{L}_{split}$. So not only we accommodate bound variables and ψ's in our lists, but also we include free variables. Those lists moreover become denumerably infinite.

Now, here is $\lfloor \cdot; \cdot; \cdot \rfloor_e$, the extended definition of the semantics of λ- and δ-items.

Definition 5.38 *(Extended λ- and δ-semantics)*
We define $\lfloor \cdot; \cdot; \cdot \rfloor_e : \mathcal{L}_{split} \times \mathcal{L}_\infty(\updownarrow) \times \Omega_\Xi^{\lambda\delta\sigma\varphi} \mapsto \overline{\Lambda}$, *such that:*
For all $t_1, t_2 \in \Omega_\Xi^{\lambda\delta}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, n \in I\!P$,

$$
\begin{array}{ll}
\lfloor \overline{v}; \overline{v'}; (t_1\lambda)t_2 \rfloor_e & =_{df} (\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e \lambda_X) \lfloor \overline{v} + +X; \overline{v'}_{\geq i+1}; t_2 \rfloor_e \textit{ for } i = nl(t_1)+1, X = hd^i(\overline{v'}) \\
\lfloor \overline{v}; \overline{v'}; (t_1\delta)t_2 \rfloor_e & =_{df} (\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e \delta) \lfloor \overline{v}; \overline{v'}_{\geq i}; t_2 \rfloor_e \textit{ for } i = nl(t_1)+1 \\
\lfloor \overline{v}; \overline{v'}; n \rfloor_e & =_{df} comp_n(\overline{v}) \\
\lfloor \overline{v}; \overline{v'}; \varepsilon \rfloor_e & =_{df} \varepsilon
\end{array}
$$

The meaning of the remaining $\Omega_{\delta\lambda\sigma\varphi}$-terms will be given below.

The following lemmas will be used in what follows:

Lemma 5.39 *For all* $\overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_\infty(\updownarrow), (\overline{v} + +\theta) \cap \overline{v'} = \emptyset, \theta \in \Theta, n, m \in I\!P$ *and* $k \in I\!N$, *we have:*

1. $\lfloor \overline{v} + +\theta; \overline{v'}; 1 \rfloor_e \equiv \theta$
2. $\lfloor \overline{v}; \overline{v'}; n + k \rfloor_e \equiv \lfloor \overline{v} + +\psi^k; \overline{v'}; n \rfloor_e$
3. $\lfloor \overline{v} + +\theta; \overline{v'}; n + 1 \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; n \rfloor_e$
4. $\lfloor \mathcal{F}_{\geq m} + +\psi^k; \overline{v'}; n \rfloor_e \equiv x_{n+k+m-1}$
5. $\lfloor \overline{v}; \overline{v'}; n \rfloor_e \in \overline{v}$
6. *If* $n \neq m$ *then* $\lfloor \overline{v}; \overline{v'}; n \rfloor_e \not\equiv \lfloor \overline{v}; \overline{v'}; m \rfloor_e$

Proof: *Easy, using Lemma 5.26 and the definition of comp.* □

Lemma 5.40 *For all* $\overline{v'} \in \mathcal{L}_{split}, \overline{v} \in \mathcal{L}(\Theta \cup \{\psi\}), \overline{v''} \in \mathcal{L}_\infty(\updownarrow), (\overline{v'} + +\overline{v}) \cap \overline{v''} = \emptyset, \theta \in \Theta$ *and* $n, i \in I\!P$, *we have:*

1. *If* $n > ||\overline{v}|| \geq 0$ *then* $\lfloor \overline{v'} + +\overline{v}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v'}; \overline{v''}; n - ||\overline{v}|| \rfloor_e$
2. *If* $n > ||\overline{v}|| \geq 0$ *then* $\lfloor \overline{v'} + +\psi^i + +\overline{v}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v''}; n + i \rfloor_e$.
3. *If* $n \leq ||\overline{v}||$ *then* $\lfloor \overline{v'} + +\overline{v}; \overline{v''}; n \rfloor_e \equiv comp_n(\overline{v})$
4. $\lfloor \overline{v'} + +\theta + +\psi + +\overline{v}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v''}; n \rfloor_e$

Proof: *This is an obvious corollary of Lemma 5.27.* □

Corollary 5.41 *For all* $\overline{v'} \in \mathcal{L}_{split}$, $\overline{v''} \in \mathcal{L}_\infty(\updownarrow), (\overline{v'} + +\overline{v}) \cap \overline{v''} = \emptyset$, *and* $n, i \in I\!P$, *we have for* $\overline{v} \in \mathcal{L}(\Theta)$:

1. *If* $n > |\overline{v}|$ *then* $\lfloor \overline{v'} + +\overline{v}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v'}; \overline{v''}; n - |\overline{v}| \rfloor_e$
2. *If* $n > |\overline{v}|$ *then* $\lfloor \overline{v'} + +\psi^i + +\overline{v}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v''}; n + i \rfloor_e$.
3. *If* $n \leq |\overline{v}|$ *then* $\lfloor \overline{v'} + +\overline{v}; \overline{v''}; n \rfloor_e \equiv comp_n(\overline{v})$

Proof: *Obvious by Lemmas 5.21 and 5.40.* □

Remark 5.42 Note that if $\overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}(\Theta \cup \{\psi\}), \overline{v''} \in \mathcal{L}_\infty(\updownarrow), (\overline{v'} + +\overline{v}) \cap \overline{v''} = \emptyset, n, i \in I\!P, ||\overline{v'}|| < 0$, then even though $n > ||\overline{v'}||$, it is not necessarily the case that:

1. $\lfloor \overline{v} + +\overline{v'}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v}; \overline{v''}; n - ||\overline{v'}|| \rfloor_e$
2. $\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; n \rfloor_e \equiv \lfloor \overline{v} + +\overline{v'}; \overline{v''}; n + i \rfloor_e$

This can be seen as follows:
$\lfloor \mathcal{F} + +\psi^5 x'; \updownarrow_{\geq 2}; 1 \rfloor_e \equiv x'$ whereas $\lfloor \mathcal{F}; \updownarrow_{\geq 2}; 1 - ||\psi^5 x'|| \rfloor_e \equiv \lfloor \mathcal{F}; \updownarrow_{\geq 2}; 5 \rfloor_e \equiv x_5$.

Now the following lemma is needed to show that $\lfloor \cdot; \cdot; \cdot \rfloor_e$ is an extension of $\lfloor \cdot; \cdot; \cdot \rfloor$.

Lemma 5.43 *For all* $\overline{v} \in \mathcal{L}(\updownarrow), \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, n \in I\!P \cup \{\varepsilon\}$, $\lfloor \overline{v}; \overline{v'}; n \rfloor \equiv \lfloor \mathcal{F} + +\overline{v}; \overline{v'}; n \rfloor_e$.

Proof: *Left as an exercise.* □

Finally, here we show that $\lfloor \cdot; \cdot; \cdot \rfloor_e$ is an extension of $\lfloor \cdot; \cdot; \cdot \rfloor$.

Lemma 5.44 *For all* $\overline{v} \in \mathcal{L}(\updownarrow), \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, t \in \Omega_\Xi^{\lambda\delta}$, $\lfloor \overline{v}; \overline{v'}; t \rfloor \equiv \lfloor \mathcal{F} + +\overline{v}; \overline{v'}; t \rfloor_e$.

Proof: *By induction on* t, *using Lemma 5.43.* □

5.5 The semantics of σ- and φ-terms

Definition 5.45 *(σ-semantics)*
For all $t_1, t_2 \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_{\infty}(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, i \in I\!P$ *we define*

$$\lfloor \overline{v}; \overline{v'}; (t_1\sigma^{(i)})t_2 \rfloor_e =_{df} \lfloor \overline{v}; \overline{v'}; t_2 \rfloor_e [\lfloor \overline{v}; \overline{v'}; i \rfloor_e := \lfloor \overline{v}; \overline{v'}_{\geq 1+nl(t_2)}; t_1 \rfloor_e]'$$

where $t_1[v := t_2]'$ *is the substitution in the mid-level given in Definition 5.4.*

Definition 5.46 *(φ-semantics)*
For all $t \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}(\Theta), \overline{v''} \in \mathcal{L}_{\infty}(\updownarrow), (\overline{v} + +\theta) \cap \overline{v''} = \emptyset, \theta \in \Theta, i \in I\!P, k \in I\!N$, *we have:*

$$\begin{array}{ll}
\lfloor \overline{v}; \overline{v''}; (\varphi^{(k,i)})t \rfloor_e & =_{df} \lfloor \overline{v}; \emptyset; \overline{v''}; (\varphi^{(k,i)})t \rfloor \\
\lfloor \overline{v}; \overline{v'}; \overline{v''}; (\varphi^{(0,i)})t \rfloor & =_{df} \lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; t \rfloor_e \\
\lfloor \overline{v} + +\theta; \overline{v'}; \overline{v''}; (\varphi^{(k+1,i)}t \rfloor & =_{df} \lfloor \overline{v}; \theta + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})t \rfloor \\
\lfloor \overline{v} + +\theta + +\psi^{k+1}; \overline{v'}; \overline{v''}; t \rfloor & =_{df} \lfloor \overline{v} + +\psi^k; \overline{v'}; \overline{v''}; t \rfloor
\end{array}$$

Note here that $\overline{v''}$ does not play a role because we do not have bound variables that we are trying to replace by variable names. What the $\overline{v'}$ does however is to save the first k variables of $\overline{v}$ which are actually the variables in t which should not be updated because they are $\leq k$. Once the first k variables of $\overline{v}$ have been saved in $\overline{v'}$, we remove the first i variables from the resulting $\overline{v}$. Hence in the end, we get the correct list from which we find the meaning of t.

Example 5.47

$$\begin{array}{lll}
1. & \lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; (\varphi^{(2,3)})3 \rfloor_e & = \lfloor \mathcal{F} + +x'; \emptyset; \updownarrow_{\geq 2}; (\varphi^{(2,3)})3 \rfloor \\
 & & = \lfloor \mathcal{F}; x'; \updownarrow_{\geq 2}; (\varphi^{(1,3)})3 \rfloor \\
 & & = \lfloor \mathcal{F}_{\geq 2}; x_1 + +x'; \updownarrow_{\geq 2}; (\varphi^{(0,3)})3 \rfloor \\
 & & = \lfloor \mathcal{F}_{\geq 2} + +\psi^3 + +x_1 + +x'; \updownarrow_{\geq 2}; 3 \rfloor_e \\
 & & = x_5 \\
2. & \lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; (\varphi^{(2,3)})1 \rfloor_e & = x' \\
3. & \lfloor \mathcal{F}; \updownarrow_{\geq 2}; (\varphi^{(1,2)})(\varphi^{(0,1)})1 \rfloor & = x_4
\end{array}$$

Now the following lemma is basic about φ-items.

Lemma 5.48 *For all* $t \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}$, $\overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}(\Theta), \overline{v''} \in \mathcal{L}_{\infty}(\updownarrow), (\overline{v} + +\overline{v'}) \cap \overline{v''} = \emptyset$ *and* $i \in I\!P$, *we have:*

$$\lfloor \overline{v} + +\overline{v'}; \overline{v''}; (\varphi^{(|\overline{v'}|,i)})t \rfloor_e \equiv \lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; t \rfloor_e$$

Proof: *Easy. First prove by induction on* $|\overline{v'}|$ *that if* $\overline{v} \in \mathcal{L}_{split}, \overline{v'}, \overline{v_1} \in \mathcal{L}(\Theta)$ *such that* $(\overline{v} + +\overline{v'} + +\overline{v_1}) \cap \overline{v''} = \emptyset$ *then*

$$\lfloor \overline{v} + +\overline{v'}; \overline{v_1}; \overline{v''}; (\varphi^{(|\overline{v'}|,i)})t \rfloor \equiv \lfloor \overline{v}; \overline{v'} + +\overline{v_1}; \overline{v''}; (\varphi^{(0,i)})t \rfloor$$

□

The following lemma opens the road to working with lists which do not contain ψ.

Lemma 5.49 *For all* $\overline{v'} \in \mathcal{L}_{split}, \overline{v} \in \mathcal{L}(\Theta \cup \{\psi\}), \overline{v_1} \in \mathcal{L}_\infty(\updownarrow), (\overline{v'} + +\theta + +\overline{v}) \cap \overline{v_1} = \emptyset, \theta \in \Theta$ *and* $n \in I\!P$, *we have:*

$$\lfloor \overline{v'} + +\theta + +\psi + +\overline{v}; \overline{v_1}; t \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v_1}; t \rfloor_e$$

Proof: *By nested induction. We prove by induction on* t *that* $IH_1(t)$ *holds where* $IH_1(t)$ *is:*

$$\lfloor \overline{v'} + +\theta + +\psi + +\overline{v}; \overline{v_1}; t \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v_1}; t \rfloor_e$$

- *Case* $t = n$, *use case 4 of lemma 5.40.*
- *Case* $(t_1\delta)t_2$ *or* $(t_1\lambda)t_2$ *or* $(t_1\sigma^{(i)})t_2$ *where* $IH_1(t_1)$ *and* $IH_1(t_2)$ *hold, easy.*
- *Case* $(\varphi^{(k,i)})t$ *where* $IH_1(t)$ *holds, prove by induction on* k *that* $IH_2(k)$ *holds where* $IH_2(k)$, *for all* $\overline{v''} \in \mathcal{L}(\Theta)$ *is:*

  $$\lfloor \overline{v'} + +\theta + +\psi + +\overline{v}; \overline{v''}; \overline{v_1}; (\varphi^{(k,i)})t \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v''}; \overline{v_1}; (\varphi^{(k,i)})t \rfloor_e$$

  - *case* $k = 0$, *use* $IH_1(t)$.
  - *Assume* $IH_2(k)$. *Now, prove by induction on* $|\overline{v}|$ *that* $IH_3(\overline{v})$ *holds where* $IH_3(\overline{v})$ *is:*

    $$\lfloor \overline{v'} + +\theta + +\psi + +\overline{v}; \overline{v''}; \overline{v_1}; (\varphi^{(k+1,i)})t \rfloor_e \equiv \lfloor \overline{v'} + +\overline{v}; \overline{v''}; \overline{v_1}; (\varphi^{(k+1,i)})t \rfloor_e$$

    * *case* $|\overline{v}| = 0$, *use Definition 5.46.*
    * *Case* $\overline{v} + +\theta$ *where* $\theta \in \Theta$ *and* $IH_3(\overline{v})$ *holds, use Definition 5.46 and* $IH_2(k)$.
    * *Case* $\overline{v} + +\theta + +\psi^j$ *where* $\theta \in \Theta, j \in I\!P$ *and* $IH_3(\overline{v} + +\psi^{j-1})$ *holds, use Definition 5.46 and* $IH_3(\overline{v} + +\psi^{j-1})$.
    * *Case* ψ^j *where* $j \in I\!P$, *use Definition 5.46.*

□

Now this lemma is very important. It says that all the ψ's can be removed from lists.

Lemma 5.50 *For all* $\overline{v} \in \mathcal{L}_{split}, \exists \overline{v'} \in \mathcal{L}_{split}$ *which is free for* ψ *such that for all* $t \in \Omega_\Xi^{\lambda\delta\sigma\varphi}, \overline{v''} \in \mathcal{L}_\infty(\updownarrow)$ *such that* $\overline{v} \cap \overline{v''} = \emptyset, \lfloor \overline{v}; \overline{v''}; t \rfloor_e \equiv \lfloor \overline{v'}; \overline{v''}; t \rfloor_e$.

Proof: *We can write* $\overline{v}$ *as* $\overline{v_1} + +\theta + +\overline{v_2}$ *such that* $\theta \in \Theta, \overline{v_1} \in \mathcal{L}_{split}, \overline{v_2} \in \mathcal{L}(\Theta \cup \{\psi\})$, $\overline{v_1}$ *is free of* ψ *and* $\overline{v_2}$ *has* ψ *as its leftmost element. Now, the proof is by induction on* $|\overline{v_2}|$ *using Lemma 5.49. Note moreover, that* $\overline{v'}$ *is independent of* t. *Hence, we may assume from now on that our start lists do not contain* ψ. □

Finally, we give the translation of any term t of $\Omega_\Xi^{\lambda\delta\sigma\varphi}$:

Definition 5.51 *(The semantic function)*
We define $\lfloor \cdot \rfloor : \Omega_\Xi^{\lambda\delta\sigma\varphi} \mapsto \overline{\Lambda}$ *such that for all* t *in* $\Omega_\Xi^{\lambda\delta\sigma\varphi}, \lfloor t \rfloor =_{df} \lfloor \mathcal{F}; \updownarrow; t \rfloor_e$

Lemma 5.52 $\lfloor \cdot \rfloor$ *is well defined. That is, for all* $t \in \Omega_\Xi^{\lambda\delta\sigma\varphi}$, $\lfloor t \rfloor$ *is a unique term in* $\overline{\Lambda}$.

Proof: *By induction on* $t \in \Omega_\Xi^{\lambda\delta\sigma\varphi}$. □

Now this is our first lemma towards the correctness of our semantics:

Lemma 5.53 *For all $t \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}$, we have:*

1. *$BV(\lfloor\overline{v};\overline{v'};t\rfloor) \subset \overline{v'}$ for every $\overline{v} \in \mathcal{L}_{split}$ and $\overline{v'} \in \mathcal{L}_{\infty}(\updownarrow)$ such that $\overline{v} \cap \overline{v'} = \emptyset$.*
2. *$FV(\lfloor\overline{v};\overline{v'};t\rfloor) \subset \overline{v}$ for every $\overline{v} \in \mathcal{L}_{split}$ and $\overline{v'} \in \mathcal{L}_{\infty}(\updownarrow)$ such that $\overline{v} \cap \overline{v'} = \emptyset$.*
3. *$BV(\lfloor t\rfloor) \subset \updownarrow$ and $FV(\lfloor t\rfloor) \subset \mathcal{F}$.*

Proof: *1 and 2 are by induction on t. 3 is a corollary of 1 and 2.* □

What this lemma means is that the term $\lfloor t\rfloor$ in $\overline{\Lambda}$ can be translated using Definition 5.6 to a term in Λ.

Let us give now a few examples:

Example 5.54 (Note that we sometimes combine many steps in one.)

$$\begin{aligned}
\lfloor(\varphi^{(2,1)})(1\delta)(2\lambda)3\rfloor &\equiv \lfloor\mathcal{F};\updownarrow;(\varphi^{(2,1)})(1\delta)(2\lambda)3\rfloor_e \\
&\equiv \lfloor\mathcal{F};\emptyset;\updownarrow;(\varphi^{(2,1)})(1\delta)(2\lambda)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 2};x_1;\updownarrow;(\varphi^{(1,1)})(1\delta)(2\lambda)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 3};x_2 + +x_1;\updownarrow;(\varphi^{(0,1)})(1\delta)(2\lambda)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 3} + +\psi + +x_2 + +x_1;\updownarrow;(1\delta)(2\lambda)3\rfloor_e \\
&\equiv (x_1\delta)(x_2\lambda_{x'})x_4
\end{aligned}$$

$$\begin{aligned}
\lfloor(\varphi^{(2,3)})(\varphi^{(1,2)})(1\delta)(2\delta)3\rfloor &\equiv \lfloor\mathcal{F};\updownarrow;(\varphi^{(2,3)})(\varphi^{(1,2)})(1\delta)(2\delta)3\rfloor_e \\
&\equiv \lfloor\mathcal{F}_{\geq 2};x_1;\updownarrow;(\varphi^{(1,3)})(\varphi^{(1,2)})(1\delta)(2\delta)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 3};x_2 + +x_1;\updownarrow;(\varphi^{(0,3)})(\varphi^{(1,2)})(1\delta)(2\delta)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +x_2 + +x_1;\updownarrow;(\varphi^{(1,2)})(1\delta)(2\delta)3\rfloor_e \\
&\equiv \lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +x_2;x_1;\updownarrow;(\varphi^{(0,2)})(1\delta)(2\delta)3\rfloor \\
&\equiv \lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +x_2 + +\psi^2 + +x_1;\updownarrow;(1\delta)(2\delta)3\rfloor_e \\
&\equiv (x_1\delta)(\lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +x_2 + +\psi^2 + +x_1;\updownarrow;2\rfloor_e\delta) \\
&\quad \lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +x_2 + +\psi^2 + +x_1;\updownarrow;3\rfloor_e \\
&\equiv (x_1\delta)(\lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +\psi;\uparrow;1\rfloor_c\delta)\lfloor\mathcal{F}_{\geq 3} + +\psi^3 + +\psi;\updownarrow;2\rfloor_e \\
&\equiv (x_1\delta)(\lfloor\mathcal{F}_{\geq 7};\updownarrow;1\rfloor_c\delta)\lfloor\mathcal{F}_{\geq 7};\updownarrow;2\rfloor_e \\
&\equiv (x_1\delta)(x_7\delta)x_8
\end{aligned}$$

6 The soundness of σ- and φ-reduction

In this section we will show that if $t \to t'$ where $\to$ is the result of a φ-transition or destruction rule, or of a σ-destruction rule, then $\lfloor t\rfloor \equiv \lfloor t'\rfloor$. That is, we will show that both φ and σ are sound in what concerns variable updating and substitution. We will show moreover, that if $t \to_\sigma t'$ where $\to$ is the firing of the σ-generation rule, then $\lfloor t\rfloor = \lfloor t'\rfloor$. That is, σ-generation is a form of β-conversion in our system. Furthermore, σ-transition accommodates in it α-conversion. That is, if $t \to_\sigma t'$ where $\to_\sigma$ is a σ-transition rule, then $\lfloor t\rfloor =_{\overline{\alpha}} \lfloor t'\rfloor$. For this, let us group all the definitions of the meaning of the different terms together:

Definition 6.1 *(Semantics of $\Omega_{\Xi}^{\lambda\delta\sigma\varphi}$) For all $t, t_1, t_2 \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}(\Theta), \overline{v''} \in \mathcal{L}_{\infty}(\updownarrow), (\overline{v} + +\theta) \cap \overline{v''} = \emptyset, \theta \in \Theta, i, n \in I\!P$ and $k \in I\!N$, we define:*

$M1.$ $\lfloor t \rfloor =_{df} \lfloor \mathcal{F}; \updownarrow; t \rfloor_e$

$M2.$ $\lfloor \overline{v}; \overline{v''}; \varepsilon \rfloor_e =_{df} \varepsilon$
$M3.$ $\lfloor \overline{v}; \overline{v''}; n \rfloor_e =_{df} comp_n(\overline{v})$
$M4.$ $\lfloor \overline{v}; \overline{v''}; (t_1\lambda)t_2 \rfloor =_{df} (\lfloor \overline{v}; \overline{v''}; t_1 \rfloor \lambda_X) \lfloor \overline{v} + +X; \overline{v''}_{\geq i+1}; t_2 \rfloor$ *for* $i = nl(t_1) + 1, X = hd^i(\overline{v''})$
$M5.$ $\lfloor \overline{v}; \overline{v''}; (t_1\delta)t_2 \rfloor =_{df} (\lfloor \overline{v}; \overline{v''}; t_1 \rfloor \delta) \lfloor \overline{v}; \overline{v''}_{\geq i}; t_2 \rfloor$ *for* $i = nl(t_1) + 1$
$M6.$ $\lfloor \overline{v}; \overline{v''}; (t_1\sigma^{(i)})t_2 \rfloor =_{df} \lfloor \overline{v}; \overline{v''}; t_2 \rfloor_e [\lfloor \overline{v}; \overline{v''}; i \rfloor_e := \lfloor \overline{v}; \overline{v''}_{\geq i}; t_1 \rfloor_e]'$ *for* $i = nl(t_2) + 1$
$M7.$ $\lfloor \overline{v}; \overline{v''}; (\varphi^{(k,i)})t \rfloor_e =_{df} \lfloor \overline{v}; \emptyset; \overline{v''}; (\varphi^{(k,i)})t \rfloor$

$M8.$ $\lfloor \overline{v}; \overline{v'}; \overline{v''}; (\varphi^{(0,i)})t_1 \rfloor =_{df} \lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; t \rfloor_e$
$M9.$ $\lfloor \overline{v} + +\theta; \overline{v'}; \overline{v''}; (\varphi^{(k+1,i)})t_1 \rfloor =_{df} \lfloor \overline{v}; \theta + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})t \rfloor$
$M10.$ $\lfloor \overline{v} + +\theta + +\psi^{k+1}; \overline{v'}; \overline{v''}; t \rfloor =_{df} \lfloor \overline{v} + +\psi^k; \overline{v'}; \overline{v''}; t \rfloor$

Let us furthermore recall here that $\Omega = \{\lambda, \delta, \sigma, \varphi\}$ and that Ω_{Ξ} is defined in Definition 2.21. Finally, the φ-rules are given in Definition 3.4 and the σ-rules are given in Definition 3.8. (We leave the discussion of μ till the next section.)

Now, the following lemmas inform us about the place of (α) in our system.

Lemma 6.2 *If $n \in I\!P, \overline{v} \in \mathcal{L}_{split}, \overline{v'}, \overline{v''} \in \mathcal{L}_{\infty}(\updownarrow)$ and $\overline{v} \cap \overline{v'} = \overline{v} \cap \overline{v''} = \emptyset$, then $\lfloor \overline{v}; \overline{v'}; n \rfloor_e = \lfloor \overline{v}; \overline{v''}; n \rfloor_e$.*

Proof: *Obvious.* □

Lemma 6.3 *If $t \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_{\infty}(\updownarrow)$ and $\overline{v} \cap \overline{v'} = \emptyset$, then for all $\overline{v''} \in \mathcal{L}_{\infty}(\overline{v'})$, $\lfloor \overline{v}; \overline{v'}; t \rfloor_e =_{\overline{\alpha}} \lfloor \overline{v}; \overline{v''}; t \rfloor_e$.*

Proof: *By induction on t.* □

Now we define the notions of (α-, β-) soundness:

Definition 6.4

- *We say that a reduction rule $\rightarrow$ is sound if:* $(\forall t, t', \overline{v}, \overline{v'})[t \rightarrow t' \Rightarrow \lfloor \overline{v}; \overline{v'}; t \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; t' \rfloor_e]$.
- *We say that a reduction rule $\rightarrow$ is α-sound if:*

  $(\forall t, t', \overline{v}, \overline{v'})[t \rightarrow t' \Rightarrow \lfloor \overline{v}; \overline{v'}; t \rfloor_e =_{\overline{\alpha}} \lfloor \overline{v}; \overline{v'}; t' \rfloor_e]$.
- *We say that a reduction rule $\rightarrow$ is β-sound if:*

  $(\forall t, t', \overline{v}, \overline{v'})[t \rightarrow t' \Rightarrow \lfloor \overline{v}; \overline{v'}; t \rfloor_e =_{\overline{\beta}} \lfloor \overline{v}; \overline{v'}; t' \rfloor_e]$.
- *We say that a reduction rule $\rightarrow$ is $\alpha\beta$-sound if:*

  $(\forall t, t', \overline{v}, \overline{v'})[t \rightarrow t' \Rightarrow \lfloor \overline{v}; \overline{v'}; t \rfloor_e = \lfloor \overline{v}; \overline{v'}; t' \rfloor_e]$.

Lemma 6.5 *φ-transition through a δ-item is sound. That is, for all $t_1, t_2 \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}$, $\overline{v_1} \in \mathcal{L}_{split}$, $\overline{v''} \in \mathcal{L}_\infty(\updownarrow)$, $\overline{v_1} \cap \overline{v''} = \emptyset$, $i \in I\!P$, and $k \in I\!N$, we have:*

$\lfloor \overline{v_1}; \overline{v''}; (\varphi^{(k,i)})(t_1\delta)t_2 \rfloor_e \equiv \lfloor \overline{v_1}; \overline{v''}; ((\varphi^{(k,i)})t_1\delta)(\varphi^{(k,i)})t_2 \rfloor_e$

Proof: *According to Lemma 5.50, we may assume that $\overline{v_1}$ is ψ-free. Assume moreover that $\overline{v_1} = \overline{v} + +\overline{v'}$ such that $|\overline{v'}| = k$.*

$$
\begin{array}{ll}
(\lfloor \overline{v} + +\overline{v'}; \overline{v''}; ((\varphi^{(k,i)})t_1\delta)(\varphi^{(k,i)})t_2 \rfloor_e & \equiv^{j=1+nl(t_1)} \\
(\lfloor \overline{v} + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})t_1 \rfloor_e \delta)\lfloor \overline{v} + +\overline{v'}; \overline{v''}_{\geq j}; (\varphi^{(k,i)})t_2 \rfloor_e & \equiv^{Lemma\ 5.48} \\
(\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; t_1 \rfloor_e \delta)\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}_{\geq j}; t_2 \rfloor_e & \equiv \\
\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; (t_1\delta)t_2 \rfloor_e & \equiv^{Lemma\ 5.48} \\
\lfloor \overline{v} + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})(t_1\delta)t_2 \rfloor_e &
\end{array}
$$

□

Lemma 6.6 *φ-transition through a λ-item is sound. That is, for all $t_1, t_2 \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}$, $\overline{v_1} \in \mathcal{L}_{split}$, $\overline{v''} \in \mathcal{L}_\infty(\updownarrow)$, $\overline{v_1} \cap \overline{v''} = \emptyset$, $i \in I\!P$, and $k \in I\!N$, we have:*

$\lfloor \overline{v_1}; \overline{v''}; (\varphi^{(k,i)})(t_1\lambda)t_2 \rfloor_e \equiv \lfloor \overline{v_1}; \overline{v''}; ((\varphi^{(k,i)})t_1\lambda)(\varphi^{(k+1,i)})t_2 \rfloor_e$

Proof: *Similarly to the above lemma, we may assume that $\overline{v_1}$ is ψ-free. Assume moreover that $\overline{v_1} = \overline{v} + +\overline{v'}$ such that $|\overline{v'}| = k$.*

$$
\begin{array}{ll}
(\lfloor \overline{v} + +\overline{v'}; \overline{v''}; ((\varphi^{(k,i)})t_1\lambda)(\varphi^{(k+1,i)})t_2 \rfloor_e & \equiv^{j=1+nl(t_1), X=hd^j(\overline{v''})} \\
(\lfloor \overline{v} + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})t_1 \rfloor_e \lambda_X)\lfloor \overline{v} + +\overline{v'} + +X; \overline{v''}_{\geq j+1}; (\varphi^{(k+1,i)})t_2 \rfloor_e & \equiv^{Lemma\ 5.48} \\
(\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; t_1 \rfloor_e \lambda_X)\lfloor \overline{v} + +\psi^i + +\overline{v'} + +X; \overline{v''}_{\geq j+1}; t_2 \rfloor_e & \equiv \\
\lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v''}; (t_1\lambda)t_2 \rfloor_e & \equiv^{Lemma\ 5.48} \\
\lfloor \overline{v} + +\overline{v'}; \overline{v''}; (\varphi^{(k,i)})(t_1\lambda)t_2 \rfloor_e &
\end{array}
$$

□

Lemma 6.7 *φ-destruction is sound. That is, for all $\overline{v_1} \in \mathcal{L}_{split}$, $\overline{v_2} \in \mathcal{L}_\infty(\updownarrow)$, $\overline{v_1} \cap \overline{v_2} = \emptyset$, $n, i \in I\!P$, $k \in I\!N$, we have:*

1. *If $n > k$ then $\lfloor \overline{v_1}; \overline{v_2}; (\varphi^{(k,i)})n \rfloor \equiv \lfloor \overline{v_1}; \overline{v_2}; n + i \rfloor$.*
2. *If $n \leq k$ then $\lfloor \overline{v_1}; \overline{v_2}; (\varphi^{(k,i)})n \rfloor_e \equiv \lfloor \overline{v_1}; \overline{v_2}; n \rfloor_e$.*

Proof: *Assume $\overline{v_1}$ is ψ-free and $\overline{v_1} = \overline{v} + +\overline{v'}$ such that $|\overline{v'}| = k$.*

1. $\lfloor \overline{v} + +\overline{v'}; \overline{v_2}; (\varphi^{(k,i)})n \rfloor_e \equiv^{Lemma\ 5.48} \lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v_2}; n \rfloor_e \equiv^{Corollary\ 5.41} \lfloor \overline{v} + +\overline{v'}; \overline{v_2}; n + i \rfloor_e$
2. $\lfloor \overline{v} + +\overline{v'}; \overline{v_2}; (\varphi^{(k,i)})n \rfloor_e \equiv^{Lemma\ 5.48} \lfloor \overline{v} + +\psi^i + +\overline{v'}; \overline{v_2}; n \rfloor_e \equiv^{Corollary\ 5.41}$
   $comp_n(\overline{v'}) \equiv^{Corollary\ 5.41} \lfloor \overline{v} + +\overline{v'}; \overline{v_2}; n \rfloor_e$

□

Lemma 6.8 *σ-destruction is sound. That is, for all $t \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}$, $\overline{v} \in \mathcal{L}_{split}$, $\overline{v'} \in \mathcal{L}_\infty(\updownarrow)$, $\overline{v} \cap \overline{v'} = \emptyset$, $i, j \in I\!P$, we have:*

1. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})i \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; t \rfloor_e$.

2. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})j \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; j \rfloor_e$ *if* $j \neq i$.

3. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})\varepsilon \rfloor_e \equiv \varepsilon$.

Proof:

1. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})i \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; i \rfloor_e [\lfloor \overline{v}; \overline{v'}; i \rfloor_e := \lfloor \overline{v}; \overline{v'}; t \rfloor_e]' \equiv \lfloor \overline{v}; \overline{v'}; t \rfloor_e$.

2. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})j \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; j \rfloor_e [\lfloor \overline{v}; \overline{v'}; i \rfloor_e := \lfloor \overline{v}; \overline{v'}; t \rfloor_e]' \equiv \lfloor \overline{v}; \overline{v'}; j \rfloor_e$, *as* $\lfloor \overline{v}; \overline{v'}; j \rfloor_e \neq \lfloor \overline{v}; \overline{v'}; i \rfloor_e$ *from Lemma 5.39.*

3. $\lfloor \overline{v}; \overline{v'}; (t\sigma^{(i)})\varepsilon \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; \varepsilon \rfloor_e [\lfloor \overline{v}; \overline{v'}; i \rfloor_e := \lfloor \overline{v}; \overline{v'}; t \rfloor_e]' \equiv \varepsilon$, *as* $\varepsilon \notin \overline{v}$, *for every* $\overline{v}$.

□

Lemma 6.9 *σ-transition is α-sound. That is, for all* $\overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v'} = \emptyset, i \in I\!P, t_1, t_2, t \in \Omega_\Xi^{\lambda\delta\sigma\varphi}$, *we have:*

1. $\lfloor \overline{v}; \overline{v'}; (t_1\sigma^{(i)})(t_2\lambda)t \rfloor_e =_{\overline{\alpha}} \lfloor \overline{v}; \overline{v'}; ((t_1\sigma^{(i)})t_2\lambda)((\varphi)t_1\sigma^{(i+1)})t \rfloor_e$

2. $\lfloor \overline{v}; \overline{v'}; (t_1\sigma^{(i)})(t_2\delta)t \rfloor_e =_{\overline{\alpha}} \lfloor \overline{v}; \overline{v'}; ((t_1\sigma^{(i)})t_2\lambda)(t_1\sigma^{(i)})t \rfloor_e$

Proof: *Left to the reader.* □

Theorem 6.10 *For all* $t, t' \in \Omega_\Xi^{\lambda\delta\sigma\varphi}$, *if* $t \rightarrow_r t'$ *where* r *is any σ- or φ-transition rule, or any σ- or φ-destruction rule, then* $\lfloor t \rfloor \equiv \lfloor t' \rfloor$.

Proof: *This is a corollary of Lemmas 6.5, 6.6, 6.7, 6.8 and 6.9 above.* □

The transition and destruction rules of σ and φ work like substitution and variable updating. Therefore, they should return equivalent terms. σ-generation on the other hand, accommodates in it β-reduction.

Example 6.11

$$\lfloor \mathcal{F}; \updownarrow; (2\delta)(3\lambda)1 \rfloor_e \equiv (\lfloor \mathcal{F}; \updownarrow; 2 \rfloor_e \delta)(\lfloor \mathcal{F}; \updownarrow; 3 \rfloor_e \lambda_{x'}) \lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; 1 \rfloor_e \equiv (x_2\delta)(x_3\lambda_{x'})x'$$

Moreover,

$$\begin{array}{ll}
\lfloor \mathcal{F}; \updownarrow; (2\delta)(3\lambda)((\varphi)2\sigma^{(1)})1 \rfloor_e & \equiv \\
(\lfloor \mathcal{F}; \updownarrow; 2 \rfloor_e \delta)(\lfloor \mathcal{F}; \updownarrow; 3 \rfloor_e \lambda_{x'}) \lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; ((\varphi)2\sigma^{(1)})1 \rfloor_e & \equiv \\
(\lfloor \mathcal{F}; \updownarrow; 2 \rfloor_e \delta)(\lfloor \mathcal{F}; \updownarrow; 3 \rfloor_e \lambda_{x'})(\lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; 1 \rfloor_e [\lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; 1 \rfloor_e := \lfloor \mathcal{F} + +x'; \updownarrow_{\geq 2}; (\varphi)2 \rfloor_e]' & \equiv \\
(\lfloor \mathcal{F}; \updownarrow; 2 \rfloor_e \delta)(\lfloor \mathcal{F}; \updownarrow; 3 \rfloor_e \lambda_{x'})(x'[x' := x_2]') & \equiv \\
(\lfloor \mathcal{F}; \updownarrow; 2 \rfloor_e \delta)(\lfloor \mathcal{F}; \updownarrow; 3 \rfloor_e \lambda_{x'})x_2 & \equiv \\
(x_2\delta)(x_3\lambda_{x'})x_2 &
\end{array}$$

Of course $(x_2\delta)(x_3\lambda_{x'})x'$ and $(x_2\delta)(x_3\lambda_{x'})x_2$ are not α-equivalent but are β-equivalent. In fact,

$$(x_2\delta)(x_3\lambda_{x'})x' \rightarrow_{\overline{\beta}} x_2 \text{ and } (x_2\delta)(x_3\lambda_{x'})x_2 \rightarrow_{\overline{\beta}} x_2.$$

Hence, our task is to show that if $t \rightarrow_\sigma t'$ where $\rightarrow_\sigma$ is σ-generation, then $\lfloor t \rfloor = \lfloor t' \rfloor$. This is done in the following lemma:

Lemma 6.12 *σ-generation is $\alpha\beta$-sound. That is, for all $t, t_1, t_2 \in \Omega_\Xi^{\lambda\delta\sigma\varphi}$, for all $\overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}_\infty(\updownarrow)$, such that $\overline{v} \cap \overline{v'} = \emptyset$, $\lfloor \overline{v}; \overline{v'}; (t_1\delta)(t_2\lambda)t \rfloor = \lfloor \overline{v}; \overline{v'}; (t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})t \rfloor$.*

Proof: *Let $i = 1 + nl(t_1), j = 1 + nl(t_2), X = hd^j(\overline{v}_{\geq i}), k = 1 + nl(t)$.*

$$
\begin{array}{ll}
\lfloor \overline{v}; \overline{v'}; (t_1\delta)(t_2\lambda)((\varphi)t_1\sigma^{(1)})t \rfloor & \equiv \\
(\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e\delta)(\lfloor \overline{v}; \overline{v'}_{\geq i}; t_2 \rfloor_e\lambda_X)(\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; ((\varphi)t_1\sigma^{(1)})t \rfloor_e) & \equiv \\
(\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e\delta)(\lfloor \overline{v}; \overline{v'}_{\geq i}; t_2 \rfloor_e\lambda_X)(\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e[X := \lfloor \overline{v} + +X; \overline{v'}_{\geq i+j+k}; (\varphi)t_1 \rfloor_e]') & =_{\overline{\beta}}^{5.48,\ 5.49} \\
(\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e[X := \lfloor \overline{v}; \overline{v'}_{\geq i+j+k}; t_1 \rfloor_e]'[X := \lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e]') & =_{\overline{\alpha}}^{Lemma\ 6.3} \\
(\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e[X := \lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e]'[X := \lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e]') & \equiv^{Lemma\ 5.53} \\
\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e[X := \lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e]' &
\end{array}
$$

Moreover,

$$
\begin{array}{ll}
\lfloor \overline{v}; \overline{v'}; (t_1\delta)(t_2\lambda)t \rfloor & \equiv \\
(\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e\delta)(\lfloor \overline{v}; \overline{v'}_{\geq i}; t_2 \rfloor_e\lambda_X)\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e & =_{\overline{\beta}} \\
\lfloor \overline{v} + +X; \overline{v'}_{\geq i+j}; t \rfloor_e[X := \lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e]' &
\end{array}
$$

□

7 The meaning and soundness of β-reduction

Recall from Definition 3.20 how we defined β-reduction. There β-reduction was defined as a combination of σ-, φ- and μ-reduction. Hence, as we have proved the soundness of σ- and φ-reduction, all we have left to show here is that μ-reduction is sound, where μ-reduction has been defined in Definition 3.19. In fact, this is what we will show in this section. More precisely, we will show that μ-generation is $\alpha\beta$-sound and that μ-destruction and transition are sound. Let us first define the meaning of terms with μ-leading items.

Definition 7.1 *(μ-semantics)*
If t is an $\Omega_{\lambda\delta}$-term, $\overline{v} \in \mathcal{L}^{-1}(\Theta), \overline{v'} \in \mathcal{L}(\Theta), \theta \in \Theta, \overline{v''} \in \mathcal{L}_\infty(\updownarrow), \overline{v} \cap \overline{v''} = \emptyset, i \in I\!P$ and i does not refer to any free variable of t, we define:

$$
\begin{array}{lll}
\lfloor \overline{v}; \overline{v''}; (\mu^{(i)})t \rfloor_e & \equiv & \lfloor \overline{v}; \emptyset; \overline{v''}; (\mu^{(i)})t \rfloor \\
\lfloor \overline{v}; \overline{v'}; \overline{v''}; (\mu^{(1)})t \rfloor & \equiv & \lfloor \overline{v} + +hd(\overline{v''}) + +\overline{v'}; \overline{v''}_{\geq 2}; t \rfloor_e \\
\lfloor \overline{v} + +\theta; \overline{v'}; \overline{v''}; (\mu^{(i+1)})t \rfloor & \equiv & \lfloor \overline{v}; \theta + +\overline{v'}; \overline{v''}; (\mu^{(i)})t \rfloor
\end{array}
$$

Note here that the provision "i does not refer to a free variable of t" can be assumed due to Lemma 3.22. In fact, this is the only case we need to define the semantics for. Note moreover that it is enough to take $\overline{v} \in \mathcal{L}^{-1}(\Theta)$ (see Definition 5.18), because t is an $\Omega_{\lambda\delta}$-term, so we never generate ψ's in the list $\overline{v}$.

Example 7.2

1. $\lfloor(\mu^{(1)})(2\lambda)1\rfloor \equiv$
   $\lfloor\mathcal{F};\updownarrow;(\mu^{(1)})(2\lambda)1\rfloor_e \equiv$
   $\lfloor\mathcal{F};\emptyset;\updownarrow;(\mu^{(1)})(2\lambda)1\rfloor \equiv$
   $\lfloor\mathcal{F}++x';\updownarrow_{\geq 2};(2\lambda)1\rfloor_e \equiv$
   $(\lfloor\mathcal{F}++x';\updownarrow_{\geq 2};2\rfloor_e\lambda_{x''})\lfloor\mathcal{F}++x';\updownarrow_{\geq 3};1\rfloor_e \equiv$
   $(x_1\lambda_{x''})x''$

2. $\lfloor(\mu^{(2)})(1\lambda)1\rfloor \equiv$
   $\lfloor\mathcal{F};\updownarrow;(\mu^{(2)})(1\lambda)1\rfloor_e \equiv$
   $\lfloor\mathcal{F};\emptyset;\updownarrow;(\mu^{(2)})(1\lambda)1\rfloor \equiv$
   $\lfloor\mathcal{F}_{\geq 2};x_1;\updownarrow;(\mu^{(1)})(1\lambda)1\rfloor \equiv$
   $\lfloor\mathcal{F}_{\geq 2}++x'++x_1;\updownarrow_{\geq 2};(1\lambda)1\rfloor_e \equiv$
   $(\lfloor\mathcal{F}_{\geq 2}++x'++x_1;\updownarrow_{\geq 2};1\rfloor_e\lambda_{x''})\lfloor\mathcal{F}_{\geq 2}++x'++x_1++x'';\updownarrow_{\geq 3};1\rfloor_e \equiv$
   $(x_1\lambda_{x''})x''$

Note that $\lfloor(\mu^{(1)})(1\lambda)1\rfloor$ is not allowed, since the superscript 1 refers to the free variable 1 (the first 1) in $(1\lambda)1$.

Lemma 7.3 *Let t be an $\Omega_{\lambda\delta}$-term. If λ° does not bind any variable in $(\lambda^\circ)(\lambda^1)(\lambda^2)\ldots(\lambda^k)t$, then $\forall\overline{v}\in\mathcal{L}^{-1}(\Theta),\overline{v''}\in\mathcal{L}(\Theta),\overline{v'}\in\mathcal{L}_\infty(\updownarrow),\theta,\theta'\in\Theta$, such that $(\overline{v'}++\overline{v''})\cap\overline{v'}=\emptyset,\theta,\theta'\notin\overline{v}\cup\overline{v'}\cup\overline{v''},|\overline{v''}|=k$, we have:*

$$\lfloor\overline{v}++\theta++\overline{v''};\overline{v'};t\rfloor_e\equiv\lfloor\overline{v}++\theta'++\overline{v''};\overline{v'};t\rfloor_e$$

Proof: *By induction on t using Lemmas 5.39 and 6.2.* □

Lemma 7.4 *If $(t_1\delta)(t_2\lambda)$ is void in $(t_1\delta)(t_2\lambda)t$, $i=1+nl(t_1),j=1+nl(t_2)$ then for all $\overline{v}\in\mathcal{L}^{-1}(\Theta)$, $\overline{v'}\in\mathcal{L}_\infty(\updownarrow)$, such that $\overline{v}\cap\overline{v'}=\emptyset$ and $X=hd^{i+j-1}(\overline{v'})$, $(\lfloor\overline{v};\overline{v'};t_1\rfloor_e\delta)(\lfloor\overline{v};\overline{v'}_{\geq i};t_2\rfloor_e\lambda_X)$ is void in $\lfloor\overline{v};\overline{v'};(t_1\delta)(t_2\lambda)t\rfloor_e$.*

Proof: *By induction on $\Omega_{\lambda\delta}$-terms t.* □

Lemma 7.5 *μ-generation is $\alpha\beta$-sound. That is, for all t_1,t_2,t $\Omega_{\lambda\delta}$-terms, for all $\overline{v}\in\mathcal{L}^{-1}(\Theta),\overline{v'}\in\mathcal{L}_\infty(\updownarrow)$ such that $\overline{v}\cap\overline{v'}=\emptyset$, if $(t_1\delta)(t_2\lambda)$ is void in t then: $\lfloor\overline{v};\overline{v'};(t_1\delta)(t_2\lambda)t\rfloor_e=\lfloor\overline{v};\overline{v'};(\mu^{(1)})t\rfloor_e$*

Proof: *By induction on t. Let $i=1+nl(t_1),j=1+nl(t_2),X=hd^i(\overline{v'}_{\geq j})=hd^{i+j-1}(\overline{v'})$.*

- *If $t\equiv\varepsilon$ then obvious.*
- *If $t\equiv m$ then $m>1$. Moreover, $(\lfloor\overline{v};\overline{v'};t_1\rfloor_e\delta)(\lfloor\overline{v};\overline{v'}_{\geq i};t_2\rfloor_e\lambda_X)\lfloor\overline{v}++X;\overline{v'}_{\geq i+j};m\rfloor_e\equiv$
  $(\lfloor\overline{v};\overline{v'};t_1\rfloor_e\delta)(\lfloor\overline{v};\overline{v'}_{\geq i};t_2\rfloor_e\lambda_X)\lfloor\overline{v};\overline{v'}_{\geq i+j};m-1\rfloor_e =^{Lemma\ 7.4}_{\beta}$
  $\lfloor\overline{v};\overline{v'}_{\geq i+j};m-1\rfloor_e\equiv^{Lemmas\ 5.39\ and\ 6.2}$
  $\lfloor\overline{v}++hd(\overline{v'});\overline{v'}_{\geq 2};m\rfloor_e\equiv\lfloor\overline{v};\overline{v'};(\mu^{(1)})m\rfloor_e$.

- *If* $t \equiv (t_1'\lambda)t_2'$ *then:* $\lfloor \overline{v}; \overline{v'}; (t_1\delta)(t_2\lambda)(t_1'\lambda)t_2' \rfloor_e \equiv^{k=1+nl(t_1'),X'=hd^k(\overline{v'}_{\geq i+j})}$
  $(\lfloor \overline{v}; \overline{v'}; t_1 \rfloor_e \delta)(\lfloor \overline{v}; \overline{v'}_{\geq i}; t_2 \rfloor_e \lambda_X)(\lfloor \overline{v}++X; \overline{v'}_{\geq i+j}; t_1' \rfloor_e \lambda_{X'}) \lfloor \overline{v}++X++X'; (\overline{v'}_{\geq i+j})_{\geq k+1}; t_2' \rfloor_e =^{\underline{Lemma\ 7.4}}_{\beta}$
  $\lfloor \overline{v}++X; \overline{v'}_{\geq i+j}; (t_1'\lambda)t_2' \rfloor_e =^{\underline{Lemma\ 6.3}}_{\alpha}$
  $\lfloor \overline{v}++X; \overline{v'}_{\geq 2}; (t_1'\lambda)t_2' \rfloor_e \equiv^{Lemma\ 7.3}$
  $\lfloor \overline{v}++hd(\overline{v'}); \overline{v'}_{\geq 2}; (t_1'\lambda)t_2' \rfloor_e \equiv \lfloor \overline{v}; \overline{v'}; (\mu^{(1)})(t_1'\lambda)t_2' \rfloor_e$
- *If* $t \equiv (t_1'\delta)t_2'$ *then similar.*

□

Remark 7.6 Note that μ-generation is not sound. In particular,

$$\begin{array}{lcll} \lfloor \mathcal{F}; \updownarrow; (4\delta)(\lambda)2 \rfloor_e & \equiv & (x_4\delta)(\lambda_{x'})x_1 & \text{and} \\ \lfloor \mathcal{F}; \updownarrow; (\mu^{(1)})2 \rfloor_e & \equiv & \lfloor \mathcal{F}++x'; \updownarrow_{\geq 2}; 2 \rfloor \equiv x_1 & \end{array}$$

Now $(x_4\delta)(\lambda_{x'})x_1 =_\beta x_1$ and $(x_4\delta)(\lambda_{x'})x_1 \not\equiv x_1$.

Lemma 7.7 *μ-transition is sound. That is, for all $\Omega_{\lambda\delta}$-terms t_1, t_2, for all $\overline{v} \in \mathcal{L}^{-1}(\Theta)$ and $\overline{v'''} \in \mathcal{L}_\infty(\updownarrow)$ such that $\overline{v} \cap \overline{v'''} = \emptyset$, for all $i \in I\!P$, if $i \neq$ all free variables of $(t_1\lambda)t_2, k = 1 + nl(t_1), X = hd^k(\overline{v'''})$ then:*

1. $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})(t_1\lambda)t_2 \rfloor_e \equiv (\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})t_1 \rfloor_X)\lfloor \overline{v}++X; \overline{v'''}_{\geq k+1}(\mu^{(i+1)})t_2 \rfloor_e$
2. $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})(t_1\delta)t_2 \rfloor_e \equiv (\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})t_1 \rfloor\delta)\lfloor \overline{v}; \overline{v'''}_{\geq k}; (\mu^{(i+1)})t_2 \rfloor_e$

Proof:

1. *Let* $\overline{v} = \overline{v'}++\overline{v''}$ *such that* $|\overline{v''}| = i-1$
   $(\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})t_1 \rfloor_e \lambda_X)\lfloor \overline{v}++X; \overline{v'''}_{\geq k+1}; (\mu^{(i+1)})t_2 \rfloor_e \equiv$
   $(\lfloor \overline{v'}++hd(\overline{v'''})++\overline{v''}; \overline{v'''}_{\geq 2}; t_1 \rfloor_e \lambda_X)\lfloor \overline{v'}++hd(\overline{v'''}_{\geq k+1})++\overline{v''}++X; \overline{v'''}_{\geq k+2}; t_2 \rfloor_e \equiv^{\underline{Lem\ 7.3}}$
   $\lfloor \overline{v'}++hd(\overline{v'''})++\overline{v''}; \overline{v'''}_{\geq 2}; (t_1\lambda)t_2 \rfloor_e \equiv$
   $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})(t_1\lambda)t_2 \rfloor_e$

2. *Is similar.*

□

Lemma 7.8 *μ-destruction is sound. That is, for all $\overline{v} \in \mathcal{L}^{-1}(\Theta)$ and $\overline{v'''} \in \mathcal{L}_\infty(\updownarrow)$ such that $\overline{v} \cap \overline{v'''} = \emptyset$, for all $i, m \in I\!P$, we have:*

- $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})\varepsilon \rfloor_e \equiv \varepsilon$.
- $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})m \rfloor_e \equiv \lfloor \overline{v'}++\overline{v''}; \overline{v'''}; m \rfloor_e$ *if* $m < i$.
- $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})m \rfloor_e \equiv \lfloor \overline{v'}++\overline{v''}; \overline{v'''}; m-1 \rfloor_e$ *if* $m > i$.

Proof:

- $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})\varepsilon \rfloor_e \equiv \varepsilon$, *easy.*
- $\lfloor \overline{v}; \overline{v'''}; (\mu^{(i)})m \rfloor_e \equiv \lfloor \overline{v'}++hd(\overline{v'''})++\overline{v''}; \overline{v'''}_{\geq 2}; m \rfloor_e$ *where* $\overline{v} = \overline{v'}++\overline{v''}$ *and* $|\overline{v''}| = i-1$
  - *If* $m < i$ *then* $m \leq i-1$ *and* $\lfloor \overline{v'}++hd(\overline{v'''})++\overline{v''}; \overline{v'''}_{\geq 2}; m \rfloor_e \equiv \lfloor \overline{v'}++\overline{v''}; \overline{v'''}; m \rfloor_e$.
  - *If* $m > i$ *then* $m \geq i+1$ *and* $\lfloor \overline{v'}++hd(\overline{v'''})++\overline{v''}; \overline{v'''}_{\geq 2}; m \rfloor_e \equiv \lfloor \overline{v'}++\overline{v''}; \overline{v'''}; m-1 \rfloor_e$.

□

8 Comparison and conclusions

In this paper we presented a calculus of substitution which is explicit hence mending the problem of the implicit substitution of the λ-calculus. Our calculus Ω_{Ξ} is based on a calculus Λ in which terms are written in item-notation. Moreover, Ω_{Ξ} uses de Bruijn's indices rather than variable names. We wrote our calculus in the most general way in order to apply our results to the various existing λ-calculi and type theories. In fact, the item-notation assumed in this paper has been shown to be general enough to accommodate the type free and all the systems of the Barendregt cube (see [NK 9x]). We believe that this notation has helped to define substitution explicitly and in a modular way with the other terms. Moreover, with our approach, local reduction and substitution can be accommodated very naturally, something which is difficult in the classical λ-calculus. In fact we have shown that it is enough to add one reduction rule in order to obtain local substitution.

In order to show the soundness of our calculus we provided a translation from Ω_{Ξ} into $\overline{\Lambda}$, a variant of Λ where bound variables are taken from a particular ordered list. Our translation functions are important on their own. First, it is nice to have a mechanical procedure which takes terms written with variable names and returns terms with de Bruijn's indices. Second, it is equally important and interesting to go the other way. For instance, when translating a lambda term (with de Bruijn indices) that represents some mathematical theory/proof to a lambda term with named variables, we want particular names to be used. In fact, one of the advantages of de Bruijn's indices is that α-conversion is no longer needed. Now, terms written with de Bruijn's indices are difficult to understand even for those who are familiar with them. Variable names on the other hand, clarify the term in hand but cause a lot of complications when applying reduction and substitution. If however, we order our lists of free and bound variables, then we can avoid the difficulty caused by variable names. In fact, this is what we do in this paper. We take our lists of variables to be ordered and we translate every term of Ω_{Ξ} into a term of $\overline{\Lambda}$ (i.e. using variable names) in a unique way via $|\cdot|$. When in $\overline{\Lambda}$, it is up to us to equate terms modulo α-conversion rather than being forced to do it in the translation (see Appendix A).

In order to make substitution explicit and to discuss β-reduction, we had to add three kinds of reduction rules: the φ-, σ- and μ-reductions. φ updates variables, σ substitutes terms for variables and μ decreases the indices as a result of a β-conversion which removes a λ from a term. Each kind of reduction has three rules: generation, transition and destruction. Now, substitution and reduction in $\overline{\Lambda}$ are given similarly to that of the classical calculus; i.e. implicit and global. Therefore, we show that our reduction rules actually do represent reduction and substitution in $\overline{\Lambda}$. This shows the soundness of our reduction rules. In particular, we show that σ-, μ- φ-destruction and φ-, μ-transition are sound in that if $t \to_r t'$ where r is one of these rules, then $|t| \equiv |t'|$. This is very nice because the corresponding reductions in $\overline{\Lambda}$ also return equivalent rather than α-equivalent terms. Furthermore, we show that σ-transition is α-sound in that if $t \to_{\sigma-transition} t'$ then $|t| =_{\overline{\alpha}} |t'|$. We also show that σ- and μ-generation are $\alpha\beta$-sound in that if $t \to_r t'$ where r is one of these two rules, then $|t| =_{\overline{\alpha\beta}} |t'|$. Now, we are satisfied with the result concerning β-conversion. In fact, these last two rules do actually represent β-conversion in Ω_{Ξ}. What we have been disappointed with however is that we had to use α-conversion rather than equivalence in the soundness proof of σ-transition and σ- and μ-generation. So even though we have avoided α-conversion in our translation function, it still had to be assumed in the soundness of three reduction rules. Look for example at the proof

of Lemma 7.5. When $t \equiv (t'_1\lambda)t'_2$, we had to apply Lemma 6.3 to obtain an α-equivalent term. This, we have not quite understood yet. Maybe in σ- and μ-generation and in σ-transition, α-conversion is necessary. Or maybe it is possible to complicate even more our lists of variables and our definition of the semantic functions so that α-conversion is really avoided. This is a point for further investigation. Finally, note that we did not discuss completeness because this becomes here a trivial matter. In fact, everything that can be shown in the classical λ-calculus can be shown in our own. Even better, our calculus is more expressive in that it accommodates explicit substitution whereas the classical one does not.

So to summarize, we believe that our item notation used in conjunction with de Bruijn's indices provide a precise formulation of the λ-calculus that can be used efficiently for implementation and theoretical purposes and that can generalise a whole collection of type and λ-theories. The usefulness of the notation is not discussed in this paper but the reader is referred to [NK 9x]. This notation however provides an explicit approach of substitution which is the most general up to date and which can be used to generalise other existing approaches of explicit substitution as shown in [KN 93]. Furthermore, the soundness of the explicit substitution and the resulting reductions is shown in terms of the classical notions of substitution and reductions. The translation functions between terms written with de Bruijn indices and terms written with variable names are useful and provide a detailled account of the notion of α-conversion. Finally, we believe that our account of explicit substitution is the most general and detailled up to date, from the point of view of both syntax and semantics. Here is a summary of the various existing accounts of explicit substitution that we are aware of and of their relation to our own:

[KN 93] provides an account of explicit substitution which is used to discuss local and global substitution and reduction. No semantics is provided for that account and the precision of this paper is not assumed there. The reduction rules however of the present paper are based on [KN 93] even though there, there was no μ-reduction and α-reduction was assumed. We believe that we have in this paper presented the most extensive approach of variable manipulation, substitution and reduction. Our approach can be easily and in a straightforward fashion implemented because we have carried out all the difficult work related to variables. The article [Abadi et al. 91] provides an algebraic syntax and semantics for explicit substitution where de Bruijn's indices are used. The connection with the classical λ-calculus is not investigated. Furthermore, [KN 93] has shown that the approach in [Abadi et al. 91] can be interpreted in [KN 93] and can be further simplified. [Hardin and Lévy 89] proposes confluent systems of substitution based on the study of categorical combinators yet we believe that our account is more comprehensive. [Field 90] provides an account of explicit substitution similar to that of [Abadi et al. 91] hence it can also be accommodated in our account. The master thesis of [van Horssen 92] discusses explicit substitution in the classical notation and the item notation assumed in this paper. [van Horssen 92] deduces that the item notation has advantages over the classical one. The master thesis of [Krab93] provides a semantics of the explicit substitution of Ω_Ξ which originated from our function e of this paper. [Krab93] however, ignores to order the list of bound variables which we call $\rfloor$. This makes it impossible for him to impose α-conversion. In appendix A, we will provide a semantics of substitution where all α-equivalent terms are identifiable.

9 Acknowledgements

We are grateful for the discussions with Jos Baeten, Henk Barendregt, Erik Barendsen, Inge Bethke, Tijn Borghuis, Herman Geuvers, Jeroen Krabbendam and Erik Poll, and for the helpful remarks received from them.

A An alternative semantics

In the definition of the semantic function from Ω_Ξ to $\overline{\Lambda}$, we took $\mathcal{F}$ and $\updownarrow$ which were both ordered (see Definition 6.1). This enabled us to translate every term t of Ω_Ξ in a unique term t' of $\overline{\Lambda}$ which is not equivalent to any other term in the α-equivalence class of t'. The price we had to pay is of course having to manipulate not only the list of free variables but also the list of bound ones. This is not a high price to pay if we compare with the substitution we have to manipulate if we assume a semantic function which identifies terms modulo α-conversion. Moreover, ignoring α-conversion is remaining with the essence of de Bruijn's indices and avoiding all this renaming of variables. Here is how we illustrate the point:

Look at Definition 5.29. We could use another semantic function which does not choose a particular index for the lambda, but any of the indices which has not been yet used. Here is this new definition:

Definition A.1 *(λ- and δ-semantics) For all $t_1, t_2 \in \Omega_\Xi^{\lambda\delta}, \overline{v} \in \mathcal{L}(\updownarrow), n \in I\!P \cup \{\varepsilon\}$,*

$$
\begin{array}{lll}
\lfloor \overline{v}; (t_1\lambda)t_2 \rfloor & =_{df} & (\lfloor \overline{v}; t_1 \rfloor \lambda_v)\lfloor \overline{v} + +v; t_2 \rfloor \text{ where } v \in \updownarrow \setminus \overline{v} \\
\lfloor \overline{v}; (t_1\delta)t_2 \rfloor & =_{df} & (\lfloor \overline{v}; t_1 \rfloor \delta)\lfloor \overline{v}; t_2 \rfloor \\
\lfloor \overline{v}; n \rfloor & =_{df} & \begin{cases} comp_n(\overline{v}) & \text{if } n \leq |\overline{v}| \\ x_{n-|\overline{v}|} & n > |\overline{v}| \\ \varepsilon & \text{if } n = \varepsilon \end{cases}
\end{array}
$$

Example A.2

$$
\begin{array}{ll}
\lfloor \emptyset; (\lambda)(1\lambda)(1\delta)3 \rfloor & \equiv^{X_1 \in \updownarrow, X_1 \text{ is arbitrary}} \\
(\lfloor \emptyset; \varepsilon \rfloor \lambda_{X_1})\lfloor X_1; (1\lambda)(1\delta)3 \rfloor & \equiv \\
(\varepsilon\lambda_{X_1})(\lfloor X_1; 1 \rfloor \lambda_{X_2})\lfloor X_1 X_2; (1\delta)3 \rfloor & \equiv^{X_2 \in \updownarrow, X_2 \text{ is arbitrary}, X_2 \not\equiv X_1} \\
(\varepsilon\lambda_{X_1})(comp_1(X_1)\lambda_{X_2})(\lfloor X_1 X_2; 1 \rfloor \delta)\lfloor X_1 X_2; 3 \rfloor & \equiv \\
(\varepsilon\lambda_{X_1})(X_1\lambda_{X_2})(comp_1(X_1 X_2)\delta)x_{3-|X_1 X_2|} & \equiv \\
(\varepsilon\lambda_{X_1})(X_1\lambda_{X_2})(X_2\delta)x_1 &
\end{array}
$$

We need the following definition of substitution which defines variable substitution of lists of variables.

Definition A.3 *(Substitution in lists) If $\overline{v}$ is a list of variables of $\overline{\Lambda}$, then we define $\overline{v}[v := v']'$ to be the list $\overline{v}$ but where all occurrences of v have been replaced by v'.*

Now the following lemmas are needed to show that $\lfloor \cdot; \cdot \rfloor$ is well defined.

Lemma A.4 *For any $\overline{v}, t, FV(\lfloor \overline{v}; t \rfloor) \subseteq \overline{v} \cup \mathcal{F}$.*

Proof: *By induction on t, recalling that ε is neither free nor bound.* □

Lemma A.5 *If* $X' \in \updownarrow \setminus \overline{v}, X \in \overline{v}, \overline{v} \in \mathcal{L}(\updownarrow)$ *and* $t \in \Omega^{\lambda\delta}_{\Xi}$, *then*

$$\lfloor \overline{v}; t \rfloor [X := X']' =_{\overline{\alpha}} \lfloor \overline{v}[X := X']'; t \rfloor.$$

Proof: *By induction on* $t \in \Omega^{\lambda\delta}_{\Xi}$.

1. $\lfloor \overline{v}; n \rfloor [X := X']' \equiv \lfloor \overline{v}[X := X']'; n \rfloor$ *for* $n \in I\!P \cup \{\varepsilon\}$.

2. $\lfloor \overline{v}; (t_1\delta)t_2 \rfloor [X := X']' \equiv ((\lfloor \overline{v}; t_1 \rfloor \delta) \lfloor \overline{v}; t_2 \rfloor)[X := X']' \equiv$
$(\lfloor \overline{v}; t_1 \rfloor [X := X']' \delta) \lfloor \overline{v}; t_2 \rfloor [X := X']' =^{IH}_{\overline{\alpha}}$
$(\lfloor \overline{v}[X := X']'; t_1 \rfloor \delta) \lfloor \overline{v}[X := X']'; t_2 \rfloor \equiv \lfloor \overline{v}[X := X']'; (t_1\delta)t_2 \rfloor$.

3. $\lfloor \overline{v}; (t_1\lambda)t_2 \rfloor [X := X']' \equiv^{X_1 \in \updownarrow \setminus \overline{v}, X_1 \not\equiv X'} ((\lfloor \overline{v}; t_1 \rfloor \lambda_{X_1}) \lfloor \overline{v} + +X_1; t_2 \rfloor)[X := X']' \equiv$
$(\lfloor \overline{v}; t_1 \rfloor [X := X']' \lambda_{X_1}) \lfloor \overline{v} + +X_1; t_2 \rfloor [X := X']' \equiv^{IH}$
$(\lfloor \overline{v}[X := X']'; t_1 \rfloor \lambda_{X_1}) \lfloor (\overline{v} + +X_1)[X := X']'; t_2 \rfloor \equiv$
$(\lfloor \overline{v}[X := X']'; t_1 \rfloor \lambda_{X_1}) \lfloor \overline{v}[X := X']' + +X_1; t_2 \rfloor \equiv \lfloor \overline{v}[X := X']'; (t_1\lambda)t_2 \rfloor$.

4. $\lfloor \overline{v}; (t_1\lambda)t_2 \rfloor [X := X']' \equiv^{X' \in \updownarrow \setminus \overline{v}} ((\lfloor \overline{v}; t_1 \rfloor \lambda_{X'}) \lfloor \overline{v} + +X'; t_2 \rfloor)[X := X']' \equiv^{X'' \notin FV(\lfloor \overline{v} + +X'; t_2 \rfloor)}$
$((\lfloor \overline{v}; t_1 \rfloor \lambda_{X''}) \lfloor \overline{v} + +X'; t_2 \rfloor [X' := X'']')[X := X']' =^{Lemma\ A.4, IH}_{\overline{\alpha}}$
$((\lfloor \overline{v}; t_1 \rfloor \lambda_{X''}) \lfloor \overline{v} + +X'[X' := X'']'; t_2 \rfloor)[X := X']' \equiv ((\lfloor \overline{v}; t_1 \rfloor \lambda_{X''}) \lfloor \overline{v} + +X''; t_2 \rfloor)[X := X']'$
Now, refer to case 3 above.

□

Lemma A.6 $(\lfloor \overline{v}; t_1 \rfloor \lambda_{X_1}) \lfloor \overline{v} + +X_1; t_2 \rfloor =_{\overline{\alpha}} (\lfloor \overline{v}; t_1 \rfloor \lambda_{X_2}) \lfloor \overline{v} + +X_2; t_2 \rfloor$ *for* $X_1, X_2 \in \updownarrow \setminus \overline{v}$.

Proof: *If* $X_1 = X_2$, *then nothing to prove.*
If $X_1 \neq X_2$, *then noting that* $X_2 \notin FV(\lfloor \overline{v} + +X_1; t_2 \rfloor)$ *by Lemma A.4, we get:*

$$\begin{array}{ll}
(\lfloor \overline{v}; t_1 \rfloor \lambda_{X_1}) \lfloor \overline{v} + +X_1; t_2 \rfloor & \equiv \\
(\lfloor \overline{v}; t_1 \rfloor \lambda_{X_2}) \lfloor \overline{v} + +X_1; t_2 \rfloor [X_1 := X_2]' & =^{Lemma\ A.5}_{\overline{\alpha}} \\
(\lfloor \overline{v}; t_1 \rfloor \lambda_{X_2}) \lfloor (\overline{v} + +X_1)[X_1 := X_2]'; t_2 \rfloor & \equiv^{X_1, X_2 \notin \overline{v}} \\
(\lfloor \overline{v}; t_1 \rfloor \lambda_{X_2}) \lfloor \overline{v} + +X_2; t_2 \rfloor & \equiv \\
\lfloor \overline{v}; (t_1\lambda)t_2 \rfloor &
\end{array}$$

Lemma A.7 $\lfloor \cdot; \cdot \rfloor$ *as defined in Definition A.1 is well defined. That is for all* $\overline{v}, t$, $\lfloor \overline{v}; t \rfloor$ *is unique up to* α*-conversion, (I.e. does not depend on the choice of* v *in clause 1 of Definition A.1).*

Proof: *By induction on* $t \in \Omega^{\lambda\delta}_{\Xi}$, *noting that the only interesting case is that of* $t \equiv (t_1\lambda)t_2$. *For this case, we use Lemma A.6.* □

Now compare this with the proof of Lemma 5.33. Note moreover that the versions of Lemmas 5.34 and 5.35 are:

Lemma A.8 *For all* $t \in \Omega^{\lambda\delta}_{\Xi}, c(t, \overline{s}, \updownarrow \setminus sl(\overline{s})) =_{\overline{\alpha}} \lfloor sl(\overline{s}); t \rfloor$.

Proof: *By induction on* t. □

Lemma A.9 *For all* $t \in \Omega^{\lambda\delta}_{\Xi}, e(t) =_{\overline{\alpha}} \lfloor \emptyset; t \rfloor$.

Proof: *Obvious.* □

Now the definition which replaces Definition 6.1 is the following:

Definition A.10 *(Semantics of $\Omega_{\Xi}^{\lambda\delta\sigma\varphi}$) For all $t, t_1, t_2 \in \Omega_{\Xi}^{\lambda\delta\sigma\varphi}, \overline{v} \in \mathcal{L}_{split}, \overline{v'} \in \mathcal{L}(\Theta), \theta \in \Theta, i, n \in I\!P, k \in I\!N$, we define:*

$$M1.\quad |t| =_{df} |\mathcal{F}; t|_e$$

$$\begin{array}{lll}
M2. & |\overline{v}; \varepsilon|_e & =_{df} \varepsilon \\
M3. & |\overline{v}; n|_e & =_{df} |comp_n(\overline{v}) \\
M4. & |\overline{v}; (t_1\lambda)t_2|_e & =_{df} (|\overline{v}; t_1|_e\lambda_X)|\overline{v} + +X; t_2|_e \text{ where } X \in \updownarrow \setminus \overline{v} \\
M5. & |\overline{v}; (t_1\delta)t_2|_e & =_{df} (|\overline{v}; t_1|_e\delta)|\overline{v}; t_2|_e \\
M6. & |\overline{v}; (t_1\sigma^{(i)})t_2| & =_{df} |\overline{v}; t_2|_e[|[\overline{v}; i|_e := |\overline{v}; t_1|_e]' \\
M7. & |\overline{v}; (\varphi^{(k,i)})t|_e & =_{df} |\overline{v}; \emptyset; (\varphi^{(k,i)})t|
\end{array}$$

$$\begin{array}{lll}
M8. & |\overline{v}; \overline{\overline{v'}}; (\varphi^{(0,i)})t_1| & =_{df} |\overline{v} + +\psi^i + +\overline{\overline{v'}}; t|_e \\
M9. & |\overline{v} + +\theta; \overline{v'}; (\varphi^{(k+1,i)})t_1| & =_{df} |\overline{v}; \theta + +\overline{v'}; (\varphi^{(k,i)})t| \\
M10. & |\overline{v} + +\theta + +\psi^{k+1}; \overline{v'}; t| & =_{df} |\overline{v} + +\psi^k; \overline{v'}; t|
\end{array}$$

We leave it to the reader to check the soundness of the reduction rules with respect to this definition.

References

[Abadi et al. 91] Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J.-J., (1991) Explicit substitutions, *Functional Programming 1 (4)*, 375-416.

[Barendregt 84] Barendregt, H., (1984) *Lambda Calculus: its Syntax and Semantics*, North-Holland.

[Barendregt 91] Barendregt, H., (1991) Introduction to generalised type systems, *Functional Programming 1(2)*, 125-154.

[Barendregt 92] Barendregt, H., (1992) Lambda calculi with types, *Handbook of Logic in Computer Science*, volume II, ed. Abramsky S., Gabbay D.M., Maibaum T.S.E., Oxford University Press.

[de Bruijn 70] Bruijn, N.G. de, (1970) The mathematical language AUTOMATH, its usage and some of its extensions, in: *Symposium on Automatic Demonstration, IRIA, Versailles, 1968*, Lecture Notes in Mathematics, 125, 29-61, Springer.

[de Bruijn 72] Bruijn, N.G. de, (1972) Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Indagationes Math. 34 (5)*, 381-392.

[Church 40] Church, A., (1940) A formulation of the simple theory of types, *Journal of Symbolic Logic 5*, 56-68.

[CH 88] Coquand T., and Huet G., (1988) The calculus of constructions, *Information and Computation 76*, 95-120.

[Field 90] Field, J., (1990) On laziness and optimality in lambda interpreters: tools for specification and analysis, 17^{th} *Annual Symposium on Principles of Programming Languages*, San Fransisco, 1-15.

[Hardin and Lévy 89] Hardin, Th. and Lévy, J.-J., (1989) A confluent calculus of substitutions, Lecture notes of the INRIA-ICOT symposium, Izu, Japan, November.

[van Horssen 92] Horssen, J.J. van, (1992) *Explicit substitution in two versions of typed lambda calculus*, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

[KN 93] Kamareddine, F., and Nederpelt, R.P., (1993) On stepwise explicit substitution, *International Journal of Foundations of Computer Science 3.*

[KN 9x] Kamareddine, F., and Nederpelt, R.P., (199x) *The Beauty of the Lambda Calculus*, to appear.

[Krab93] Krabbendam, J., (1993) On the soundness of explicit substitution, Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

[Nederpelt 87] Nederpelt, R.P., (1987) *De Taal van de Wiskunde*, Versluys, Almere.

[NK 9x] Nederpelt, R.P., and Kamareddine, F., (199x) A unified approach to type theory through a refined λ-calculus, paper presented at the 1992 conference on *Mathematical Foundations of Programming Semantics*, submitted for publication in the proceedings.

[NGdV 94] Nederpelt, R.P., Geuvers, J.H., and de Vrijer, R.C., eds, (1994) *Selected papers on Automath*, North-Holland, Amsterdam.

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.

92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.