# Bridging de Bruijn indices and variable names in explicit substitutions calculi[*]

Fairouz Kamareddine and Alejandro Ríos [†]

May 7, 1996

**Abstract**

Calculi of explicit substitutions have almost always been presented using de Bruijn indices with the aim of avoiding $\alpha$-conversion and being as close to machines as possible. De Bruijn indices however, though very suitable for the machine, are difficult to human users. This is the reason for a renewed interest in systems of explicit substitutions using variable names. Formal systems of explicit substitutions using variable names is a new area however and we believe, it should not develop without being well-tied to existing work on explicit substitutions. The aim of this paper is to establish a bridge between explicit substitutions using de Bruijn indices and using variable names. In our aim to do so, we provide the $\lambda t$-calculus: a $\lambda$-calculus à la de Bruijn which can be translated into a $\lambda$-calculus with explicit substitutions written with variables names. We present explicitly this translation and use it to obtain preservation of strong normalisation for $\lambda t$. Moreover, we show several properties of $\lambda t$, including confluence on closed terms and efficiency to simulate $\beta$-reduction.

## 1 Introduction

The classical $\lambda$-calculus deals with substitution in an implicit way. This means that the computations to perform substitution are usually described with operators which do not belong to the language of the $\lambda$-calculus. There has however been an interest in formalising substitution explicitly; several calculi including new operators to denote substitution and new rules to handle these operators have been proposed. Amongst these calculi we mention $C\lambda\xi\phi$ (cf. [dB78b]); the calculi of categorical combinators (cf. [Cur86]); $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\sigma_{SP}$ (cf. [ACCL91], [CHL92], [Río93]) referred to as the $\lambda\sigma$-family; $\varphi\sigma BLT$ (cf. [KN93]); $\lambda\upsilon$ (cf. [BBLRD95]) and $\lambda\zeta$ (cf. [MH95]) which are descendants of the $\lambda\sigma$-family; $\lambda s$ (cf. [KR95a]) and $\lambda s_e$ (cf. [KR96]).

All the calculi above mentioned are described in de Bruijn notation (cf. [dB72] and [dB78a]). This formalism consists in replacing the usual variable names with natural numbers which account for the bindings of the variables they stand for. This notation is useful because, while avoiding the problem of clashes of name variables, and therefore the use of Barendregt's convention and $\alpha$-congruence, it provides term rewriting systems instead of just abstract rewriting systems and therefore more rewriting tools are available to study them. The only

[*]This work was carried out under EPSRC grant GR/K25014.

[†]Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland, fax: +44 41 330 4913, *email*: fairouz@dcs.gla.ac.uk and rios@dcs.gla.ac.uk

inconvenience is that the terms written in de Bruijn notation are more suitable to be read by a computing device than by humans.

Recently, a simple calculus with explicit substitutions, $\lambda\mathbf{exp}$, has been introduced (cf. [Blo95]). This calculus is written in the standard notation with variable names and enjoys the property of Preservation of Strong Normalisation (PSN). This property states that every term that is strongly normalising (i.e. does not admit an infinite reduction path) in the classical $\lambda$-calculus is also strongly normalising in the $\lambda\mathbf{exp}$-calculus. The interest in studying such a property relies on its connection with the strong normalisation of typed calculi and the fact that several calculi of explicit substitutions do not enjoy it, as shown in [Mel95]. As a matter of fact, of the above mentioned calculi only $\lambda\upsilon$, $\lambda s$ and $\lambda\zeta$ have PSN.

The following question poses itself :*is the $\lambda\mathbf{exp}$-calculus equivalent to one of the already known calculi in de Bruijn notation, and, if not, can we describe $\lambda\mathbf{exp}$ in de Bruijn notation in a satisfactory manner?* Trying to answer this question we realized that $\lambda s$, which intuitively[1] was the best candidate for a de Bruijn version of $\lambda\mathbf{exp}$, was not the answer. Thus we were led to a new calculus, which we call $\lambda t$, whose formulation is slighty different from the formulation of $\lambda s$ and whose relationship with $\lambda\mathbf{exp}$ can be, partly, explained.

Although the rules of $\lambda t$ and $\lambda s$ are similar, both calculi work quite differently: while $\lambda s$ makes global updatings just before performing a substitution, the $\lambda t$-calculus makes partial updatings so that the computation of the updating is already finished before substitution. These partial updatings are started every time a substitution must be applied to an abstraction. Since the calculi of the $\lambda\sigma$-family, $\lambda\upsilon$ and $\lambda\zeta$ also introduce an updating operator when evaluating substitutions within abstractions, the $\lambda t$-caculus can be considered as a calculus written in the $\lambda s$-style which works with the updating mechanism of the $\lambda\sigma$-calculi and therefore as a calculus that links both $\lambda s$ and $\lambda\sigma$ styles.

In this paper we introduce $\lambda t$, we prove its confluence using the "interpretation method" ([Har89], [CHL92]), we make explicit the relationship between $\lambda t$ and $\lambda\mathbf{exp}$, which happens to be a sort of inmersion, and we use this inmersion to prove the PSN for $\lambda t$ using the PSN of $\lambda\mathbf{exp}$. We compare $\lambda t$ with $\lambda\sigma$ by providing an inmersion of the former into the latter and argue about the impossibility of such an inmersion into $\lambda\upsilon$. We also prove that $\lambda t$ is more efficient (the reductions paths are shorter) to simulate $\beta$-reduction than $\lambda\upsilon$, which seems to be the most efficient of the calculi in the $\lambda\sigma$-style. Finally, we discuss the problem of extending $\lambda t$ to a confluent calculus on open terms (terms which may contain term variables) and show that the existence of such an extension seems impossible. We conclude by explaining the problems found when trying to establish an inmersion of $\lambda\mathbf{exp}$ into $\lambda t$.

## 2 Preliminaries

We begin by presenting the notation and recalling the main notions concerning rewriting. Then we give a quick presentation of the $\lambda$-calculus à la de Bruijn. We recall afterwards the $\lambda\mathbf{exp}$-calculus and its PSN property. We explicit the isomorphism between the classical $\lambda$-calculus and its de Bruijn version. Finally, we recall the $\lambda s$-calculus so that the reader could compare it to the $\lambda t$-caculus to be introduced in section 4.

[1]Our intuition relied on the fact that both $\lambda\mathbf{exp}$ and $\lambda s$ possess an infinity of substitutions operators and that $\lambda\mathbf{exp}$ is a "minimal" extension of the classical $\lambda$-calculus "as" $\lambda s$ is of the $\lambda$-caculus à la de Bruijn

## 2.1 Rewriting

We begin by introducing the notation we shall use throughout this paper concerning rewriting and we recall the definitions of the essential properties of the reduction systems.

**Definition 1** *Let $A$ be a set and $R$ a binary relation on $A$, i.e. a subset of $A \times A$. We denote the fact $(a,b) \in R$ by $a \rightarrow_R b$ or $a \rightarrow b$ when the context is clear enough. We call* reduction *this relation and* reduction system*, the pair $(A,R)$. We denote $\overset{=}{\rightarrow}_R$ the reflexive closure of $R$. We denote $\twoheadrightarrow_R$ or just $\twoheadrightarrow$ the reflexive and transitive closure of $R$. When $a \twoheadrightarrow b$ we say there exists a* derivation *from $a$ to $b$. By $a \twoheadrightarrow^n b$, we mean that the derivation consists of $n$ steps of reduction and call $n$ the* length *of the derivation.*

**Definition 2** *Let $R$ be a reduction on $A$.*

1. *$R$ is* locally confluent *or WCR (*weakly Church-Rosser*) when*

$$\forall a,b,c \in A \ \exists d \in A \ ((a \rightarrow b \wedge a \rightarrow c) \Rightarrow (b \twoheadrightarrow d \wedge c \twoheadrightarrow d)).$$

2. *$R$ is* confluent *or CR (*Church-Rosser*) when*

$$\forall a,b,c \in A \ \exists d \in A \ ((a \twoheadrightarrow b \wedge a \twoheadrightarrow c) \Rightarrow (b \twoheadrightarrow d \wedge c \twoheadrightarrow d)).$$

**Definition 3** *Let $R$ be a reduction on $A$.*

*We say that $a \in A$ is an $R$-*normal form *(R-nf for short) if there exists no $b \in A$ such that $a \rightarrow b$ and we say that $b$* has a normal form *if there exists a normal form $a$ such that $b \twoheadrightarrow a$.*

*$R$ is* strongly normalising *or SN if there is no infinite sequence $(a_i)_{i \geq 0}$ in $A$ such that $a_i \rightarrow a_{i+1}$ for all $i \geq 0$.*

**Remark 1** *Confluence of $R$ guarantees unicity of $R$-normal forms and SN ensures their existence. When there exists a unique $R$-normal form of a term $a$, it is denoted by $R(a)$.*

## 2.2 The classical $\lambda$-calculus in de Bruijn notation

We assume the reader familiar with de Bruijn notation. Let us just say here that de Bruijn indices (or numbers) are used to make the bindings explicit: to find the $\lambda$ which binds a variable represented by the number $\mathtt{n}$ you must travel upwards in the tree associated with the term and choose the $n$-th $\lambda$ you find. For instance, $\lambda x.\lambda y.xy$ is written using de Bruijn indices as $\lambda\lambda(\mathtt{2}\,\mathtt{1})$ and $\lambda x.\lambda y.(x(\lambda z.zx))y$ is written as $\lambda\lambda(\mathtt{2}(\lambda(\mathtt{1}\,\mathtt{3}))\mathtt{1})$. Finally, to translate free variables, you must assume a fixed ordered list of binders and prefix the term to be translated with this list. For instance, if the list (written from left to right) is $\cdots, \lambda z, \lambda y, \lambda x$ then the term $\lambda x.yz$ translates as $\lambda\mathtt{3}\,\mathtt{4}$ whereas $\lambda x.zy$ translates as $\lambda\mathtt{4}\,\mathtt{3}$. The translations between both notations will be given explicitly in Section 2.4.

The interest in introducing de Bruijn indices is that they avoid clashes of variable names and therefore neither $\alpha$-conversion nor Barendregt's convention are needed. Here is the $\lambda$-calculus à la de Bruijn.

**Definition 4** *We define $\Lambda$, the* set of terms with de Bruijn indices*, as follows:*

$$\Lambda ::= \mathbb{N} \mid (\Lambda\Lambda) \mid (\lambda\Lambda)$$

*We use $a, b, \ldots$ to range over $\Lambda$ and $m, n, \ldots$ to range over $\mathbb{N}$ (positive natural numbers). Throughout the whole article, $a = b$ is used to mean that $a$ and $b$ are syntactically identical.*

*We say that a reduction $\to$ is* compatible on $\Lambda$ *when for all $a$, $b$, $c \in \Lambda$, we have $a \to b$ implies $a\,c \to b\,c$, $c\,a \to c\,b$ and $\lambda a \to \lambda b$.*

We assume the usual conventions about parentheses and avoid them when no confusion occurs. Furthermore, they shall be omitted in the grammars to be defined later.

In order to define $\beta$-reduction à la de Bruijn, we must define the substitution of a variable $\mathtt{n}$ for a term $b$ in a term $a$. Therefore, we must identify amongst the numbers of the term $a$ those that correspond to the variable $\mathtt{n}$. Furthermore, we need to update the term $b$ (rename its variables) in order to preserve the correct bindings after the replacement of the variable by $b$.

For example, translating $(\lambda x \lambda y.zxy)(\lambda x.yx) \to_\beta \lambda u.z(\lambda x.yx)u$ to de Bruijn notation we get $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \to_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$. But if we simply replace $\mathtt{2}$ in $\lambda\mathtt{521}$ by $\lambda\mathtt{31}$ we get $\lambda\mathtt{5}(\lambda\mathtt{31})\mathtt{1}$, which is not correct. We needed to decrease $\mathtt{5}$ as one $\lambda$ disappeared and to increment the free variables of $\lambda\mathtt{31}$ as they occur within the scope of one more $\lambda$.

For incrementing the free variables we need a family of updating functions:

**Definition 5** *The* updating functions $U^i_k : \Lambda \to \Lambda$ *for $k \geq 0$ and $i \geq 1$ are defined inductively as follows:*

$$U^i_k(ab) = U^i_k(a)\, U^i_k(b) \qquad U^i_k(\mathtt{n}) = \begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k\,. \end{cases}$$
$$U^i_k(\lambda a) = \lambda(U^i_{k+1}(a))$$

The intuition behind $U^i_k$ is the following: $k$ tests for free variables and $i-1$ is the value by which a variable, if free, must be incremented.

Now we define the family of meta-substitution functions:

**Definition 6** *The* meta-substitutions at level $i$, *for $i \geq 1$, of a term $b \in \Lambda$ in a term $a \in \Lambda$, denoted $a\{\!\{\mathtt{i} \leftarrow b\}\!\}$, is defined inductively on $a$ as follows:*

$$(a_1a_2)\{\!\{\mathtt{i} \leftarrow b\}\!\} = (a_1\{\!\{\mathtt{i} \leftarrow b\}\!\})\,(a_2\{\!\{\mathtt{i} \leftarrow b\}\!\}) \qquad \mathtt{n}\{\!\{\mathtt{i} \leftarrow b\}\!\} = \begin{cases} \mathtt{n-1} & \text{if } n > i \\ U^i_0(b) & \text{if } n = i \\ \mathtt{n} & \text{if } n < i\,. \end{cases}$$
$$(\lambda a)\{\!\{\mathtt{i} \leftarrow b\}\!\} = \lambda(a\{\!\{\mathtt{i+1} \leftarrow b\}\!\})$$

Ultimately, the intention is to define $(\lambda a)b \to_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$ (see definition 7 below). The first two equalities propagate the substitution through applications and abstractions and the last one carries out the substitution of the intended variable (when $n = i$) by the updated term. If the variable is not the intended one it must be decreased by 1 if it is free (case $n > i$) beacuse one $\lambda$ has disappeared, whereas if it is bound (case $n < i$) it must remain unaltered.

It is easy to check that $(\lambda\mathtt{521})\{\!\{\mathtt{1} \leftarrow (\lambda\mathtt{31})\}\!\} = \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$. This will mean $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \to_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$, as expected.

The following lemmas establish the properties of the meta-substitutions and updating functions. The Meta-substitution and Distribution lemmas are crucial to prove the confluence of $\lambda s$. The proofs of lemmas 1 - 6 are obtained by induction on $a$. Furthermore, the proof of lemma 3 requires lemma 2 with $p = 0$; the proof of lemma 4 uses lemmas 1 and 3 both with $k = 0$; finally, lemma 5 with $p = 0$ is needed to prove lemma 6.

**Lemma 1** *For* $k < n \leq k+i$ *we have:* $U^i_k(a) = U^{i+1}_k(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$ .

**Lemma 2** *For* $p \leq k < j+p$ *we have:* $U^i_k(U^j_p(a)) = U^{j+i-1}_p(a)$ .

**Lemma 3** *For* $k+i \leq n$ *we have:* $U^i_k(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U^i_k(a\{\!\{\mathtt{n}-\mathtt{i}+\mathtt{1} \leftarrow b\}\!\})$ .

**Lemma 4 (Meta-substitution lemma)** *For* $1 \leq i \leq n$ *we have:*

$$a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n}+\mathtt{1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n}-\mathtt{i}+\mathtt{1} \leftarrow c\}\!\}\}\!\}$$

**Lemma 5** *For* $p+j \leq k+1$ *we have:* $U^i_k(U^j_p(a)) = U^j_p(U^i_{k+1-j}(a))$ .

**Lemma 6 (Distribution lemma)** *For* $n \leq k+1$ *we have:*

$$U^i_k(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U^i_{k+1}(a)\{\!\{\mathtt{n} \leftarrow U^i_{k-n+1}(b)\}\!\} .$$

**Definition 7** *The* $\lambda$-calculus à la de Bruijn *is the reduction system* $(\Lambda, \rightarrow_\beta)$ *where* $\rightarrow_\beta$ *is the least compatible reduction on* $\Lambda$ *generated by the single rule:*

*($\beta$-rule)* $\qquad (\lambda a)\, b \rightarrow_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$

Finally, the following lemma ensures the good passage of the $\beta$-rule through the meta-substitutions and the $U^i_k$.

**Lemma 7** *Let* $a$, $b$, $c$, $d \in \Lambda$.

1. *If* $c \rightarrow_\beta d$ *then* $U^i_k(c) \rightarrow_\beta U^i_k(d)$ .
2. *If* $c \rightarrow_\beta d$ *then* $a\{\!\{\mathtt{i} \leftarrow c\}\!\} \twoheadrightarrow_\beta a\{\!\{\mathtt{i} \leftarrow d\}\!\}$ .
3. *If* $a \rightarrow_\beta b$ *then* $a\{\!\{\mathtt{i} \leftarrow c\}\!\} \rightarrow_\beta b\{\!\{\mathtt{i} \leftarrow c\}\!\}$ .

**Proof:** The first item is proved by induction on $c$. We just check the interesting case which arises when $c = c_1c_2$ and the reduction takes place at the root, i.e. $c_1 = (\lambda a)$, $c_2 = b$ and $d = a\{\!\{\mathtt{1} \leftarrow b\}\!\}$:

$$U^i_k((\lambda a)b) = (\lambda(U^i_{k+1}(a)))U^i_k(b) \rightarrow_\beta U^i_{k+1}(a)\{\!\{\mathtt{1} \leftarrow U^i_k(b)\}\!\} \overset{L\,6}{=} U^i_k(a\{\!\{\mathtt{1} \leftarrow b\}\!\})$$

The second item is proved by induction on $a$ using 1 above.

The third item is also proved by induction on $a$. For the case $a = (\lambda d)e$ and $b = d\{\!\{\mathtt{1} \leftarrow e\}\!\}$, Lemma 4 is required. □

This lemma was used in [KR95a] to prove the confluence of $\lambda s$. We shall only use in this paper the first item. Nevertheless we have included here the complete version in order that the reader could compare these results with the analogous results for the new meta-substitutions and updatings which shall be introduced in section 3.

In order to define the set of free variables of a term in de Bruijn notation we need first to define the following operations on sets of natural numbers.

**Definition 8** *Let* $N \subset \mathbb{N}$ *and* $k \geq 0$. *We define*

1. $N \setminus k = \{n-k : n \in N, n > k\}$ , $N + k = \{n+k : n \in N\}$
2. $N_{>k} = \{n \in N : n > k\}$ , $N_{<k} = \{n \in N : n < k\}$

3. $N_{\geq k} = \{n \in N : n \geq k\}$, $N_{\leq k} = \{n \in N : n \leq k\}$.

The following properties of the operations defined above will be needed later and their proofs are easy.

**Remark 2** *Let $N$, $M \subset \mathbb{N}$ and $k$, $k' \geq 0$. We have*

1. $(N \cup M) \setminus k = (N \setminus k) \cup (M \setminus k)$, $(N \cup M) + k = (N + k) \cup (M + k)$.
2. $(N \setminus k) \setminus k' = N \setminus (k + k')$.
3. $N \setminus 1 = N_{>1} \setminus 1$, $(N_{>k+1} \setminus 1) + 1 = (N_{>k+1} + 1) \setminus 1$.
4. $(N + k) \setminus 1 = N + (k - 1)$ *if* $k \geq 1$.
5. $(N \setminus 1)_{<k} = (N_{<k+1}) \setminus 1$, $(N \setminus 1)_{\leq k} = (N_{\leq k+1}) \setminus 1$.
6. $(N \setminus 1)_{>k} = (N_{>k+1}) \setminus 1$, $(N \setminus 1)_{\geq k} = (N_{\geq k+1}) \setminus 1$

We can define now the free variables of a term in $\Lambda$.

**Definition 9** *The* set of free variables *of a term in $\Lambda$ is defined by induction as follows:*

$$FV(\mathtt{n}) = \{n\}$$
$$FV(a\,b) = FV(a) \cup FV(b)$$
$$FV(\lambda a) = FV(a) \setminus 1$$

**Lemma 8** *For $a \in \Lambda$ we have $FV(U^i_k(a)) \setminus k = (FV(a) \setminus k) + (i - 1)$.*

**Proof:** Induction on $a$. Use Remark 2.1 for the case $a = b\,c$ and Remark 2.2 for the case $a = \lambda b$. □

**Lemma 9** *For $a$, $b \in \Lambda$ and $j \geq 1$, the following hold:*

1. $FV(a\{\!\{\mathtt{j} \leftarrow b\}\!\}) = (FV(a))_{<j} \cup ((FV(a))_{>j} \setminus 1)$ *if* $j \notin FV(a)$.
2. $FV(a\{\!\{\mathtt{j} \leftarrow b\}\!\}) = (FV(a))_{<j} \cup ((FV(a))_{>j} \setminus 1) \cup (FV(b) + (i - 1))$ *if* $j \in FV(a)$.

**Proof:** By simultaneous induction on $a$. Use the previous lemma for the case $a = \mathtt{j}$ and Remark 2.4, 5, 6 for the case $a = \lambda b$. □

**Lemma 10** *If $a \to_\beta b$ then $FV(b) \subseteq FV(a)$.*

**Proof:** By induction on $a$. The interesting case is when $a$ is an application and the contraction takes place at the root. The previous lemma settles this case. □

## 2.3 The $\lambda$-calculus and the $\lambda_{\mathbf{exp}}$-calculus

We assume the reader familiar with the $\lambda$-calculus (cf. [Bar84]) in classical notation. We just recall the syntax of its terms and the definition of $\beta$-reduction.

**Definition 10** *Given a set of variables $V = \{v_n : n \in \mathbb{N}\}$ we define recursively the set of terms $\Lambda_V$ as follows:*

$$\Lambda_V ::= V \mid \Lambda_V \, \Lambda_V \mid \lambda V.\Lambda_V$$

*We use $x$, $y, \ldots$ to range over $V$ and $A$, $B, \ldots$ to range over $\lambda_V$. We assume that different variable names stand for different variables.*

*We say that a reduction $\to$ is* compatible on $\Lambda_V$ *when for all $A$, $B$, $C \in \Lambda_V$ and $x \in V$, we have $A \to B$ implies $A\,C \to B\,C$, $C\,A \to C\,B$ and $\lambda x.A \to \lambda x.B$.*

*Barendregt's variable convention (see [Bar84]), abbreviated VC, is used and $\alpha$-congruent terms (terms which only differ on the name of bound variables) are identified.*

The classical notions of meta-substitution and $\alpha$-congruence are defined as usual (cf. [Bar84]). The meta-substitution of $x$ by $B$ in $A$ is denoted by $A[x := B]$ and $A \equiv B$ means that $A$ and $B$ are $\alpha$-congruent.

**Definition 11** *The* $\lambda$-calculus *is the reduction system $(\Lambda_V, \to_\lambda)$, where $\to_\lambda$ is the least compatible reduction on $\Lambda_V$ generated by:*

*($\beta$-rule)* $\qquad (\lambda x.A)\, B \to A[x := B]$

The $\lambda_{\mathbf{exp}}$-calculus of [Blo95] is a calculus of explicit substitutions where variable names are used instead of de Bruijn numbers. Its set of rules is minimal and the rule of *substitution-abstraction-transition* mimicks the definition of the meta-substitution acting with an abstraction. The $\lambda_{\mathbf{exp}}$-calculus is defined in [Blo95] in item notation (cf. [KN96]), but, since we are not going to exploit here the advantages of this notation, we present its standard form.

We begin by giving the syntax of the terms:

**Definition 12** *Given a set of variables $V = \{v_n : n \in \mathbb{N}\}$ we define recursively the set of terms $\Lambda\mathbf{exp}$ as follows:*

$$\Lambda\mathbf{exp} ::= V \mid \Lambda\mathbf{exp}\, \Lambda\mathbf{exp} \mid \lambda V.\Lambda\mathbf{exp} \mid \Lambda\mathbf{exp}\, \sigma_V\, \Lambda\mathbf{exp}$$

*We use $x$, $y, \ldots$ to range over $V$ and $A$, $B, \ldots$ to range over $\Lambda\mathbf{exp}$. We assume that different variable names stand for different variables. We call the terms which do not contain $\sigma$'s,* pure terms *and identify them with the terms of the classical $\lambda$-calculus.*

*We say that a reduction $\to$ is* compatible on $\Lambda\mathbf{exp}$ *when for all $A$, $B$, $C \in \Lambda\mathbf{exp}$ and $x \in V$, we have $A \to B$ implies $A\,C \to B\,C$, $C\,A \to C\,B$, $\lambda x.A \to \lambda x.B$, $A\sigma_x C \to B\sigma_x C$ and $C\sigma_x A \to C\sigma_x B$.*

*A trivially extended Barendregt's variable convention is used and $\alpha$-congruent terms (see below) are identified.*

**Definition 13** *The* set of free variables *of a term $A$, denoted $FV(A)$, the* meta-substitution *of $x$ by $B$ in a term $A$, denoted $A[x := B]$, and the notion of* $\alpha$-congruence *between terms $A$ and $B$, denoted $A \equiv B$ are defined as usual, with the respective extra clauses:*

*1.* $FV(C\sigma_x D) = (FV(C) - \{x\}) \cup FV(D)$

*2.* $(C\sigma_x D)[x := E] = C\sigma_x(D[x := E])$
$(C\sigma_x D)[y := E] = (C[y := E])\sigma_x(D[y := E])$ *with* $x \notin FV(E)$ *or* $y \notin FV(C)$

*3.* $C\sigma_x D \equiv C[x := y]\sigma_y D$

**Definition 14** *The* $\lambda\mathbf{exp}$-calculus *is the reduction system* $(\Lambda\mathbf{exp}, \rightarrow_{\lambda\mathbf{exp}})$*, where* $\rightarrow_{\lambda\mathbf{exp}}$ *is the least compatible reduction on* $\Lambda\mathbf{exp}$ *generated by the rules given below:*

| | | | | |
|---|---|---|---|---|
| *σ-generation* | $(\lambda x.A)\, B$ | $\longrightarrow$ | $A\,\sigma_x\, B$ | |
| *σ-λ-transition* | $(\lambda y.A)\,\sigma_x B$ | $\longrightarrow$ | $\lambda y.(A\,\sigma_x B)$ | *(*)* |
| *σ-app-transition* | $(A\,B)\,\sigma_x C$ | $\longrightarrow$ | $(A\,\sigma_x C)\,(B\,\sigma_x C)$ | |
| *σ-var1* | $x\,\sigma_x A$ | $\longrightarrow$ | $A$ | |
| *σ-var2* | $y\,\sigma_x A$ | $\longrightarrow$ | $y$ | |

*In (*) we have the condition* $y \notin FV(B)$*, which can be assumed to hold always due to VC.*

*We use* $\lambda\mathbf{exp}$ *to denote this set of rules. The calculus of substitutions associated with the* $\lambda\mathbf{exp}$*-calculus is the rewriting system whose rules are* $\lambda\mathbf{exp} - \{\sigma\text{-generation}\}$ *and we call it* **exp**-calculus (in [Blo95] it is called $\sigma^-$).

The main result in [Blo95] is the preservation of strong normalisation of the $\lambda\mathbf{exp}$-calculus with respect to classical $\lambda$-calculus:

**Theorem 1 (PSN of $\lambda\mathbf{exp}$)** *Every term which is strongly normalising in the classical* $\lambda$*-calculus is also strongly normalising in the* $\lambda\mathbf{exp}$*-calculus.*

## 2.4 Isomorphism between $(\Lambda_V, \rightarrow_\lambda)$ and $(\Lambda, \rightarrow_\beta)$

It is well known that the classical $\lambda$-calculus and its de Bruijn version are isomorphic rewriting systems. Nevertheless we explicit here the isomorphism, since we are going to extend it later.

**Definition 15** *For every term* $A \in \Lambda_V$ *such that* $FV(A) \subseteq \{x_1, \ldots, x_n\}$ *we define, by induction on* $A$*,* $w_{[x_1,\ldots,x_n]}(A)$ *as follows:*

$$w_{[x_1,\ldots,x_n]}(v_i) = \min\{j : v_i = x_j\}$$
$$w_{[x_1,\ldots,x_n]}(BC) = w_{[x_1,\ldots,x_n]}(B)w_{[x_1,\ldots,x_n]}(C)$$
$$w_{[x_1,\ldots,x_n]}(\lambda x.B) = \lambda w_{[x,x_1,\ldots,x_n]}(B)$$

*The notation* $[x_1, \ldots, x_n]$ *stands for the ordered list whose elements are* $x_1, \ldots, x_n$*.*

Remark that the previous definition is correct, i.e. that $\alpha$-congruent terms have the same image. This is a consequence of the following lemma.

**Lemma 11** *Let* $A \in \Lambda_V$ *such that* $FV(A) \subseteq \{x_1, \ldots, x_n\}$ *and let* $y \notin \{x_1, \ldots, x_n\}$*. Then* $w_{[x_1,\ldots,x_n]}(A) = w_{[x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n]}(A[x_i := y])$*.*

**Proof:** Easy induction on $A$. $\square$

We define now a uniform $w$, i.e. not depending on the free variables of the term.

**Definition 16** *Let $\{v_1, \dots, v_n, \dots\}$ be an enumeration of $V$. We define $w : \Lambda_V \to \Lambda$ as the function given by $w(A) = w_{[v_1,\dots,v_n]}(A)$ where $n$ is such that $FV(A) \subseteq \{v_1, \dots, v_n\}$.*

The definition is correct in the following sense.

**Lemma 12** *Let $A \in \Lambda_V$ such that $FV(A) \subseteq \{x_1, \dots, x_n\}$ and let $y_1, \dots, y_m$ be arbitrary variables. Then $w_{[x_1,\dots,x_n,y_1,\dots,y_m]}(A) = w_{[x_1,\dots,x_n]}(A)$.*

**Proof:** Easy induction on $A$. $\square$

We need to establish some lemmas before proving that $w$ preserves reduction. These lemmas state how the functions $w_{[x_1,\dots,x_n]}$ behave with the updating functions and the meta-substitutions.

**Lemma 13** *Let $A \in \Lambda_V$, $k \geq 0$, $i \geq 1$ and $n \geq k+i$ such that $x_{k+1}, \dots, x_{k+i-1} \notin FV(A)$. Then $w_{[x_1,\dots,x_n]}(A) = U^i_k(w_{[x_1,\dots,x_k,x_{k+i},\dots,x_n]}(A))$.*

**Proof:** By induction on $A$. The case $A = a\,b$ only needs the inductive hypothesis (IH). Therefore, we just study:

$A = v_m$ **:** Let $j = \min\{i : v_m = x_i\}$. Then $w_{[x_1,\dots,x_n]}(v_m) = j$.

If $j \leq k$ we have $w_{[x_1,\dots,x_n]}(A) = \mathtt{j} = U^i_k(\mathtt{j}) = U^i_k(w_{[x_1,\dots,x_k,x_{k+i},\dots,x_n]}(A))$.

If $j \geq k+i$ we have $w_{[x_1,\dots,x_n]}(A) = \mathtt{j} = U^i_k(\mathtt{j} - \mathtt{i} + \mathtt{1}) = U^i_k(w_{[x_1,\dots,x_k,x_{k+i},\dots,x_n]}(A))$.

$A = \lambda x.B$ **:** We have $w_{[x_1,\dots,x_n]}(A) = \lambda w_{[x,x_1,\dots,x_n]}(B) \overset{IH}{=} \lambda U^i_{k+1}(w_{[x,x_1,\dots,x_k,x_{k+i},\dots,x_n]}(B)) = U^i_k(\lambda(w_{[x,x_1,\dots,x_k,x_{k+i},\dots,x_n]}(B)) = U^i_k(w_{[x_1,\dots,x_k,x_{k+i},\dots,x_n]}(A))$. $\square$

**Lemma 14** *Let $A$, $B \in \Lambda_V$ such that the bound variables of $B$ are not free in $A$ and let $i \geq 1$, $\overline{y} = y_1, \dots, y_{i-1}$ and $\overline{x} = x_1, \dots, x_n$ such that $x$ is not bound in $B$, $x$ is distinct from $y_1, \dots, y_{i-1}$ and $y_1, \dots, y_{i-1} \notin FV(A)$. Then $w_{[\overline{y},\overline{x}]}(B[x := A]) = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$.*

**Proof:** By induction on $B$. We just study the interesting cases:

$B = z \in V$ **:** If $z = x$, then

$$w_{[\overline{y},\overline{x}]}(B[x := A]) = w_{[\overline{y},\overline{x}]}(A) \overset{L13}{=} U^i_0(w_{[\overline{x}]}(A)) = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$$

If $\{j : z = y_j\} \neq \phi$, let $k = \min\{j : z = y_j\}$. Then

$$w_{[\overline{y},\overline{x}]}(B[x := A]) = \mathtt{k} = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$$

If $\{j : z = x_j\} \neq \phi$, let $k = \min\{j : z = x_j\}$. We can assume $x_k \neq x$ since the case $z = x$ has already been considered. We have

$$w_{[\overline{y},\overline{x}]}(B[x := A]) = \mathtt{k} + \mathtt{i} - \mathtt{1} = \mathtt{k} + \mathtt{i}\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\} = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$$

$B = \lambda z.D$ **:** Remark that, since $x$ is not bound in B, $x \neq z$. We have

$$w_{[\overline{y},\overline{x}]}(B[x := A]) = \lambda w_{[z,\overline{y},\overline{x}]}(D[x := A]) \overset{IH}{=} \lambda(w_{[z,\overline{y},x,\overline{x}]}(D))\{\!\{\mathtt{i} + \mathtt{1} \leftarrow w_{[\overline{x}]}(A)\}\!\} =$$

$$(\lambda w_{[z,\overline{y},x,\overline{x}]}(D))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\} = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$$

Remark that we were able to apply the IH because, by VC, $z \notin FV(A)$. □

**Theorem 2** *Let $A$, $B \in \Lambda_V$, if $A \to_\lambda B$ then $w(A) \to_\beta w(B)$.*

**Proof:** It is enough to show that if $FV(A) \subseteq \{x_1, \ldots, x_n\}$ then $w_{[x_1,\ldots,x_n]}(A) \to_\beta w_{[x_1,\ldots,x_n]}(B)$. Remark that since $FV(B) \subseteq FV(A)$ (cf. [Bar84]), $w_{[x_1,\ldots,x_n]}(B)$ is well defined.

The proof is by induction on $A$. The interesting case is when A is an application and the reduction takes place at the root.

Therefore, let $A = (\lambda x.D)E$ and $B = D[x := E]$. We have

$$w_{[x_1,\ldots,x_n]}(A) = (\lambda w_{[x,x_1,\ldots,x_n]}(D))w_{[x_1,\ldots,x_n]}(E) \to_\beta$$

$$(w_{[x,x_1,\ldots,x_n]}(D))\{\!\{\mathtt{1} \leftarrow w_{[x_1,\ldots,x_n]}(E)\}\!\} \overset{L\,14}{=} w_{[x_1,\ldots,x_n]}(D[x := E]) = w_{[x_1,\ldots,x_n]}(B)$$

Remark that the conditions on the variables of Lemma 14 hold thanks to VC. □

We give now the inverse of $w$:

**Definition 17** *Let $a \in \Lambda t$ such that $FV(a) \subseteq \{1, \ldots, n\}$ and let $x_1, \ldots, x_n \in V$. We define $u_{[x_1,\ldots,x_n]}(d)$ by induction on $d$ as follows:*

$$u_{[x_1,\ldots,x_n]}(\mathtt{i}) = x_i$$
$$u_{[x_1,\ldots,x_n]}(a\,b) = u_{[x_1,\ldots,x_n]}(a)u_{[x_1,\ldots,x_n]}(b)$$
$$u_{[x_1,\ldots,x_n]}(\lambda b) = \lambda x.u_{[x,x_1,\ldots,x_n]}(b) \quad \textit{with } x \notin \{x_1, \ldots, x_n\}$$

In order to check that Definition 17 is correct, we must verify that $FV(a) \subseteq \{1, \ldots, n+1\}$ whenever $FV(\lambda.a) \subseteq \{1, \ldots, n\}$, which is obvious, and also that the definition of $u_{[x_1,\ldots,x_n]}$ on abstractions does not depend on the choice of the variable $x$. This proof is analogous to the proof of Lemma 29 and Lemma 30, which state the results we need for an extension of $u$.

We remark that we have defined for each $a \in \Lambda$ a translation into $\Lambda_V$ which depends on $n$ where $n$ is such that $FV(a) \subseteq \{1, \ldots, n\}$. We remove now this condition and define a uniform translation on $\Lambda$.

**Definition 18** *Let $\{v_1, \ldots, v_n, \ldots\}$ be the same enumeration of $V$ as in Definition 16, we define $u : \Lambda \to \Lambda_V$ as the function given by $u(a) = u_{[v_1,\ldots,v_n]}(a)$ where $n$ is such that $FV(a) \subseteq \{1, \ldots, n\}$.*

The definition is correct thanks to Lemma 31 below, which generalizes the result we need to an extension of $u$.

As we did for $w$ we can also check that $u$ preserves classical reduction and to achieve this we must establish some lemmas which make the interaction of $u$ with the updating and meta-substitutions functions precise. Since these lemmas will not be used later, we include them here for the sake of completeness and we just state them without giving detailed proofs.

**Lemma 15** *Let $a \in \Lambda$, $i \geq 1$, $k \geq 0$ and $n \geq k+i$ such that $FV(a) \subseteq \{1, \ldots, n-i+1\}$. Then $u_{[x_1,\ldots,x_n]}(U_k^i(a)) \equiv u_{[x_1,\ldots,x_k,x_{k+i},\ldots,x_n]}(a)$.*

**Lemma 16** *Let $a$, $b \in \Lambda$ and $x_1, \ldots, x_n$, $y_1, \ldots, y_{i-1}$, $x$ distinct variables.*
*Then $u_{[y_1,\ldots,y_{i-1},x_1,\ldots,x_n]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) \equiv (u_{[y_1,\ldots,y_{i-1},x,x_1,\ldots,x_n]}(a))[x := u_{[x_1,\ldots,x_n]}(b)]$.*

**Theorem 3** *Let $a$, $b \in \Lambda$, if $a \rightarrow_\beta b$ then $u(a) \rightarrow_\lambda u(b)$.*

We must only check now that in some sense $w \circ u = Id$ and $u \circ w = Id$. We begin by studying $w \circ u$, which as expected is exactly the identity.

**Lemma 17** *For every $a \in \Lambda$ we have $w(u(a)) = a$.*

**Proof:** It is enouh to show that if $FV(a) \subseteq \{1, \ldots, n\}$ then $w_{[x_1,\ldots,x_n]}(u_{[x_1,\ldots,x_n]}(a)) = a$.
This is done by induction on $a$. The usual two interesting cases are:

$a = \mathtt{i}$ : Since $x_1, \ldots, x_n$ are distinct variables, we have: $w_{[x_1,\ldots,x_n]}(u_{[x_1,\ldots,x_n]}(a)) = w_{[x_1,\ldots,x_n]}(u_{[x_1,\ldots,x_n]}(\mathtt{i})) = w_{[x_1,\ldots,x_n]}(x_i) = \mathtt{i} = a$.

$a = \lambda b$ : We have: $w_{[x_1,\ldots,x_n]}(u_{[x_1,\ldots,x_n]}(a)) = w_{[x_1,\ldots,x_n]}(\lambda x.u_{[x,x_1,\ldots,x_n]}(b) = \lambda w_{[x,x_1,\ldots,x_n]}(u_{[x,x_1,\ldots,x_n]}(b)) \overset{IH}{=} \lambda b$. □

As expected, we will not be able to obtain $u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) = A$, but we have $\alpha$-equivalence: $u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) \equiv A$.

**Lemma 18** *For every $A \in \Lambda_V$ we have $u(w(A)) \equiv A$.*

**Proof:** By induction on $A$. □

The following corollary is an immediate consequence of the two previous lemmas.

**Corollary 1** *The classical $\lambda$-calculus $(\Lambda_V, \rightarrow_\lambda)$ and the $\lambda$-calculus à la de Bruijn $(\Lambda, \rightarrow_\beta)$ are isomorphic.*

**Theorem 4** *The $\lambda$-calculus à la de Bruijn is confluent.*

**Proof:** The confluence of the classical $\lambda$-calculus (cf. [Bar84] thm. 3.2.8) is transportable, via the isomorphism, to the $\lambda$-calculus à la de Bruijn.
A proof which does not use the mentioned isomorphism is given in [Río93] (Corollary 3.6) as a corollary of a more general result concerning the $\lambda\sigma$-calculus. □

## 2.5 The $\lambda s$-calculus

We end this section by recalling the $\lambda s$-calculus and reminding the origin of its rules. We shall follow the same intuition to formulate the rules of the $\lambda t$-calculus.

The idea is to handle explicitly the meta-operators defined in definitions 5 and 6. Therefore, the syntax of the $\lambda s$-calculus is obtained by adding to the syntax of the $\lambda$-calculus à la de Bruijn two families of operators :

- $\{\sigma^i\}_{i\geq 1}$ : this family is meant to denote the explicit substitution operators. Each $\sigma^i$ is an infix operator of arity 2 and $a\,\sigma^i b$ has as intuitive meaning the term $a$ where all free occurrences of the variable corresponding to the de Bruijn number $i$ are to be substituted by the term $b$.
- $\{\varphi^i_k\}_{k\geq 0\ i\geq 1}$ : this family is meant to denote the updating functions necessary when working with de Bruijn numbers to fix the variables of the term to be substituted.

**Definition 19** *The set of* terms of the $\lambda s$-calculus*, denoted* $\Lambda s$ *, is given as follows:*

$$\Lambda s ::= \mathbb{N} \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s\, \sigma^i \Lambda s \mid \varphi^i_k \Lambda s \qquad where \quad i \geq 1\,,\ \ k \geq 0\,.$$

*We take $a$, $b$, $c$ to range over $\Lambda s$. A term of the form $a\,\sigma^i b$ is called a* closure*. Furthermore, a term containing neither $\sigma$'s nor $\varphi$'s is called a* pure term.

The $\lambda s$-calculus should carry out, besides $\beta$-reduction, the computations of updating and substitution explicitly. For that reason it contains, besides the rule mimicking the $\beta$-rule (*$\sigma$-generation*), a set of rules which are the equations in definitions 5 and 6 oriented from left to right.

**Definition 20** *The* $\lambda s$-calculus *is the reduction system $(\Lambda s, \rightarrow_{\lambda s})$, where $\rightarrow_{\lambda s}$ is the least compatible reduction on $\Lambda s$ generated by the rules given below:*

| | | | |
|---|---|---|---|
| *$\sigma$-generation* | $(\lambda a)\, b$ | $\longrightarrow$ | $a\,\sigma^1 b$ |
| *$\sigma$-$\lambda$-transition* | $(\lambda a)\, \sigma^i b$ | $\longrightarrow$ | $\lambda(a\,\sigma^{i+1} b)$ |
| *$\sigma$-app-transition* | $(a_1\, a_2)\, \sigma^i b$ | $\longrightarrow$ | $(a_1\,\sigma^i b)\,(a_2\,\sigma^i b)$ |
| *$\sigma$-destruction* | $\mathtt{n}\,\sigma^i b$ | $\longrightarrow$ | $\begin{cases} \mathtt{n-1} & \textit{if}\ \ n > i \\ \varphi^i_0\, b & \textit{if}\ \ n = i \\ \mathtt{n} & \textit{if}\ \ n < i \end{cases}$ |
| *$\varphi$-$\lambda$-transition* | $\varphi^i_k(\lambda a)$ | $\longrightarrow$ | $\lambda(\varphi^i_{k+1}\, a)$ |
| *$\varphi$-app-transition* | $\varphi^i_k(a_1\, a_2)$ | $\longrightarrow$ | $(\varphi^i_k\, a_1)\,(\varphi^i_k\, a_2)$ |
| *$\varphi$-destruction* | $\varphi^i_k\, \mathtt{n}$ | $\longrightarrow$ | $\begin{cases} \mathtt{n+i-1} & \textit{if}\ \ n > k \\ \mathtt{n} & \textit{if}\ \ n \leq k \end{cases}$ |

*We use $\lambda s$ to denote this set of rules. The calculus of substitutions associated with the $\lambda s$-calculus is the rewriting system whose rules are $\lambda s - \{\sigma\text{-generation}\}$ and we call it* $s$-calculus.

The main results concerning the $\lambda s$-calculus are (see [KR95a] for proofs):

**Theorem 5** *The $\lambda s$-calculus is confluent on $\Lambda s$.*

**Theorem 6 (PSN of $\lambda s$)** *Every $\lambda$-term which is strongly normalising in the classical $\lambda$-calculus is also strongly normalising in the $\lambda s$-calculus.*

## 3 Another presentation of the $\lambda$-calculus à la de Bruijn

In Definition 6 we have defined $\mathtt{i}\{\!\{\mathtt{i} \leftarrow b\}\!\} = U_0^i(b)$, but there is another choice: instead of updating $b$ just before performing the substitution we can make partial updatings of $b$, each time the substitution operator traverses a $\lambda$ in order to have a term already updated and simplify the equality by introducing a new meta-substitution $[\![\ \leftarrow\ ]\!]$ such that $\mathtt{i}[\![\mathtt{i} \leftarrow b]\!] = b$. Of course this simplification is only apparent since the definition of the substitution applied to an abstraction will become more involved. With these ideas in mind we propose the following definitions:

**Definition 21** *The* new updating functions $V_k : \Lambda \to \Lambda$ *for* $k \geq 0$ *are defined inductively as follows (compare with Definition 5):*

$$V_k(ab) = V_k(a)\, V_k(b) \qquad V_k(\mathtt{n}) = \begin{cases} \mathtt{n+1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k\,. \end{cases}$$
$$V_k(\lambda a) = \lambda(V_{k+1}(a))$$

**Definition 22** *The* new meta-substitutions at level $i$, *for* $i \geq 1$, *of a term* $b \in \Lambda$ *in a term* $a \in \Lambda$, *denoted* $a[\![\mathtt{i} \leftarrow b]\!]$, *are defined inductively on* $a$ *by (compare with Definition 6):*

$$a_1 a_2[\![\mathtt{i} \leftarrow b]\!] = (a_1[\![\mathtt{i} \leftarrow b]\!])(a_2[\![\mathtt{i} \leftarrow b]\!]) \qquad \mathtt{n}[\![\mathtt{i} \leftarrow b]\!] = \begin{cases} \mathtt{n-1} & \text{if } n > i \\ b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i\,. \end{cases}$$
$$(\lambda a)[\![\mathtt{i} \leftarrow b]\!] = \lambda(a[\![\mathtt{i+1} \leftarrow V_0(b)]\!])$$

Before studying the properties of these new functions let us establish the relationship between them and the old ones.

**Notation 1** *We denote the ith iteration of* $V_k$ *with itself by* $V_k^i$, *i.e.* $V_k^i(a) = V_k(\ldots(V_k a)\ldots)$ *(i times). By convention,* $V_k^0(a) = a$.

**Lemma 19** *For* $a$, $b \in \Lambda$, $i \geq 1$ *and* $k \geq 0$ *we have:*

1. $U_k^i(a) = V_k^{i-1}(a)$.
2. $a\{\!\{\mathtt{i} \leftarrow b\}\!\} = a[\![\mathtt{i} \leftarrow V_0^{i-1}(b)]\!]$.

**Proof:** Easy induction on the structure of $a$. □

**Remark 3** *As a particular case of Lemma 19.2 we have* $a\{\!\{\mathtt{1} \leftarrow b\}\!\} = a[\![\mathtt{1} \leftarrow b]\!]$ *and hence we can describe* $\beta$*-reduction using the new meta-substitution functions as:*

*($\beta$-rule)* $\quad (\lambda a)\, b \to_\beta a[\![\mathtt{1} \leftarrow b]\!]$

Unfortunately Lemma 19 cannot be used to prove all the properties we need to establish for the new updating and meta-substitutions functions by exploiting the properties we already know for the old functions. Nevertheless, it will work for some of them.

**Lemma 20** *For* $k \geq 0$ *we have* $V_k(V_0^k(c)) = V_0^{k+1}(c)$.

**Proof:** By Lemma 2, $U_k^2(U_0^{k+1}(c)) = U_0^{k+2}(c)$. Now, use Lemma 19.1. □

The following lemma, though related to Lemma 1, cannot be deduced directly from it, as we did for the previous lemma.

**Lemma 21** *For $i$, $k \geq 0$, we have $V_k^i(a) = V_k^{i+1}(a)[\![\mathtt{i}+\mathtt{k}+\mathtt{1} \leftarrow V_0^k(b)]\!]$.*

**Proof:** By indcution on the structure of $a$. $\square$

Again, the next lemma, though related to Lemma 3, cannot be deduced from it.

**Lemma 22** *For $n > k$, we have $V_k(a[\![\mathtt{n} \leftarrow V_0^k(c)]\!]) = V_k(a)[\![\mathtt{n}+\mathtt{1} \leftarrow V_0^{k+1}(c)]\!]$.*

**Proof:** By induction on the structure of $a$ and using Lemma 20 for the case $a = \mathtt{n}$. $\square$

We are ready to prove now the Meta-substitution Lemma for this new meta-substitution.

**Lemma 23 (New Meta-substitution Lemma)** *If $1 \leq i \leq n$, we have*

$$a[\![\mathtt{i} \leftarrow b]\!][\![\mathtt{n} \leftarrow V_0^{i-1}(c)]\!] = a[\![\mathtt{n}+\mathtt{1} \leftarrow V_0^i(c)]\!][\![\mathtt{i} \leftarrow b[\![\mathtt{n} \leftarrow V_0^{i-1}(c)]\!]]\!]$$

**Proof:** By induction on $a$. Lemma 22 is necessary for the case $a = \lambda d$ and Lemma 21 settles the case $a = \mathtt{n}+\mathtt{1}$. $\square$

Finally, the following lemma ensures the good passage of the $\beta$-rule through the new meta-substitutions and updatings. It is crucial for the proof of the confluence of $\lambda t$.

**Lemma 24** *Let $a$, $b$, $c$, $d \in \Lambda$.*

1. *If $c \to_\beta d$ then $V_k(c) \to_\beta V_k(d)$ .*
2. *If $c \to_\beta d$ then $a[\![\mathtt{i} \leftarrow c]\!] \twoheadrightarrow_\beta a[\![\mathtt{i} \leftarrow d]\!]$ .*
3. *If $a \to_\beta b$ then $a[\![\mathtt{i} \leftarrow c]\!] \to_\beta b[\![\mathtt{i} \leftarrow c]\!]$ .*

**Proof:**

1. It is a consequence of Lemma 19.1 and Lemma 7.1
2. Induction on $a$ using 1 above.
3. Induction on $a$. The interesting case is $a = (\lambda d)e$ and $b = d[\![\mathtt{1} \leftarrow e]\!]$:
   $((\lambda d)e)[\![\mathtt{i} \leftarrow c]\!] = (\lambda(d[\![\mathtt{i}+\mathtt{1} \leftarrow V_0(c)]\!]))(e[\![\mathtt{i} \leftarrow c]\!]) \to_\beta$
   $(d[\![\mathtt{i}+\mathtt{1} \leftarrow V_0(c)]\!])[\![\mathtt{1} \leftarrow e[\![\mathtt{i} \leftarrow c]\!]]\!] \overset{L\,23}{=} (d[\![\mathtt{1} \leftarrow e]\!])[\![\mathtt{i} \leftarrow c]\!]$ $\square$

## 4 The $\lambda t$-calculus

Now, we shall handle explicitly the new meta-operators defined in definitions 21 and 22. Therefore, the syntax of the $\lambda t$-calculus is obtained by adding to the syntax of the $\lambda$-calculus à la de Bruijn two families of operators :

- $\{\varsigma^i\}_{i\geq 1}$ : this family is meant to denote the explicit substitution operators. Each $\varsigma^i$ is an infix operator of arity 2 and $a\,\varsigma^i b$ has as intuitive meaning the term $a$ where all free occurrences of the variable corresponding to the de Bruijn number $i$ are to be substituted by the *already updated* term $b$.

- $\{\theta_k\}_{k\geq 0}$ : this family is meant to denote the new updating functions.

**Definition 23** *The set of* terms of the $\lambda t$-calculus, *denoted* $\Lambda t$ , *is given as follows:*

$$\Lambda t ::= \mathbb{N} \mid \Lambda t \Lambda t \mid \lambda \Lambda t \mid \Lambda t\, \varsigma^i \Lambda t \mid \theta_k \Lambda t \qquad where \quad i \geq 1\,, \quad k \geq 0\,.$$

*We take $a$, $b$, $c$ to range over $\Lambda t$. A term of the form $a\,\varsigma^i b$ is called a* closure. *Furthermore, a term containing neither $\varsigma$'s nor $\theta$'s is called a* pure term. *By $\theta^i_k a$ for $i \geq 1$, we mean $\theta_k(\theta_k(\ldots(\theta_k a)))$ ($i$ $\theta_k$-operators) and $\theta^0_k a$ means $a$.*

The $\lambda t$-calculus should carry out, as the $\lambda s$-calculus, besides $\beta$-reduction, the computations of updating and substitution explicitly. For that reason we include, besides the rule mimicking the $\beta$-rule ($\varsigma$-*generation*), a set of rules which are the equations in the definitions 21 and 22 oriented from left to right.

**Definition 24** *The* $\lambda t$-calculus *is the reduction system* $(\Lambda t, \rightarrow_{\lambda t})$, *where* $\rightarrow_{\lambda t}$ *is the least compatible reduction on* $\Lambda t$ *generated by the rules given below:*

| | | | |
|---|---|---|---|
| $\varsigma$-*generation* | $(\lambda a)\, b$ | $\longrightarrow$ | $a\,\varsigma^1\, b$ |
| $\varsigma$-$\lambda$-*transition* | $(\lambda a)\,\varsigma^i b$ | $\longrightarrow$ | $\lambda(a\,\varsigma^{i+1}\,\theta_0(b))$ |
| $\varsigma$-*app-transition* | $(a_1\, a_2)\,\varsigma^i b$ | $\longrightarrow$ | $(a_1\,\varsigma^i b)\,(a_2\,\varsigma^i b)$ |
| $\varsigma$-*destruction* | $\mathtt{n}\,\varsigma^i b$ | $\longrightarrow$ | $\begin{cases} \mathtt{n-1} & \textit{if} \;\; n > i \\ b & \textit{if} \;\; n = i \\ \mathtt{n} & \textit{if} \;\; n < i \end{cases}$ |
| $\theta$-$\lambda$-*transition* | $\theta_k(\lambda a)$ | $\longrightarrow$ | $\lambda(\theta_{k+1}\, a)$ |
| $\theta$-*app-transition* | $\theta_k(a_1\, a_2)$ | $\longrightarrow$ | $(\theta_k\, a_1)\,(\theta_k\, a_2)$ |
| $\theta$-*destruction* | $\theta_k\,\mathtt{n}$ | $\longrightarrow$ | $\begin{cases} \mathtt{n+1} & \textit{if} \;\; n > k \\ \mathtt{n} & \textit{if} \;\; n \leq k \end{cases}$ |

*We use $\lambda t$ to denote this set of rules. The calculus of substitutions associated with the $\lambda t$-calculus is the rewriting system whose rules are $\lambda t - \{\varsigma$-generation$\}$ and we call it* $t$-calculus.

The main difference between $\lambda t$ and $\lambda s$ can be summarized as follows: the $\lambda t$-calculus generates a partial updating when a substitution is evaluated on an abstraction (i.e. introduces an operator $\theta_0$ in the $\varsigma$-$\lambda$-*transition* rule) whereas the $\lambda s$-calculus produces a global updating when performing substitutions (i.e. introduces a $\varphi^i_0$ operator in the $\sigma$-*destruction* rule, case $n = i$).

The $\lambda t$-calculus shares this mechanism of partial updatings with the $\lambda\sigma$-caculi and their descendants $\lambda\upsilon$ and $\lambda\zeta$ since all of them introduce an updating operator in their *substitution-abstraction-transition* rule.

We shall prove now the confluence of the $\lambda t$-calculus. First we must establish some results concerning the associated calculus of substitutions $t$.

**Theorem 7 (SN and confluence of $t$)** *The $t$-calculus is SN and confluent on $\Lambda t$. Hence, every term $a$ has a unique $t$-normal form denoted $t(a)$.*

**Proof:** Let us define recursively two weight functions $W_1$ and $W_2$:

$$\begin{array}{ll} W_1(\mathtt{n}) = 2 & W_2(\mathtt{n}) = 1 \\ W_1(a\,b) = W_1(a) + W_1(b) & W_2(a\,b) = W_2(a) + W_2(b) + 1 \\ W_1(\lambda a) = W_1(a) + 2 & W_2(\lambda a) = W_2(a) + 1 \\ W_1(\theta_k a) = W_1(a) & W_2(\theta_k a) = 2W_2(a) \\ W_1(a\,\varsigma^i b) = W_1(a)(W_1(b)) & W_2(a\,\varsigma^i b) = W_2(a)(W_2(b) + 1) \end{array}$$

It is easy to check that for every rule $a \to b$ in $t$ we have $W_1(a) \geq W_1(b)$ and, furthermore, if the rule is $\varsigma$-$\lambda$-*transition* then $W_1(a) > W_1(b)$.

On the other hand, for every rule $a \to b$ in $t - \{\varsigma$-$\lambda$-*transition*$\}$ we have $W_2(a) > W_2(b)$.

Therefore, one can show by induction on $a$ that whenever $a \to b$, $(W_1(a), W_2(a)) >_{\mathrm{lex}} (W_1(b), W_2(b))$, where $>_{\mathrm{lex}}$ is the lexicographical order in $\mathbb{N} \times \mathbb{N}$.

Since there are no critical pairs, the theorem of Knuth-Bendix (cf. [KB70] or [Hue80]) applies trivially to yield the local confluence of the $t$-calculus.

Finally, Newman's lemma, which states that every strong normalising and locally confluent relation is confluent (cf. [Bar84], Proposition 3.1.25), provides the confluence of the $t$-calculus.

□

**Lemma 25** *The set of $t$-normal forms is exactly $\Lambda$.*

**Proof:** Check first by induction on $a$ that $a\,\varsigma^i b$ and $\theta_k a$ are not normal forms. Then check by induction on $a$ that if $a$ is a $t$-nf then $a \in \Lambda$. Conclude by observing that every term in $\Lambda$ is a $t$-nf. □

**Lemma 26** *For all $a$, $b \in \Lambda t$ we have:*

$$t(a\,b) = t(a)t(b)\ , \quad t(\lambda a) = \lambda(t(a))\ , \quad t(\theta_k a) = V_k(t(a))\ , \quad t(a\,\varsigma^i b) = t(a)[\![\mathtt{i} \leftarrow t(b)]\!]\ .$$

**Proof:** The first and second equalities are immediate since there are no $t$-rules whose left-hand side is an application or an abstraction.

Prove the third equality for terms in $t$-nf, i.e. use an inductive argument on $c \in \Lambda$ to show $t(\theta_k c) = V_k(t(c))$. Let now $a \in \Lambda t$, $t(\theta_k a) = t(\theta_k t(a)) = V_k(t(t(a))) = V_k(t(a))$.
Prove the fourth claim similarly using the third one. □

We give now the key result that allows us to use the Interpretation Method in order to get the confluence of the $\lambda t$-calculus: the good passage of the $\varsigma$-generation rule to the $t$-normal forms.

**Lemma 27** *Let $a$, $b \in \Lambda t$ , if $a \to_{\varsigma-gen} b$ then $t(a) \twoheadrightarrow_{\beta} t(b)$ .*

**Proof:** Induction on $a$. We just study the interesting cases.

$a = c\,d$ **:** If the reduction takes place within $c$ or $d$ just use the IH. The interesting case is when $c = \lambda e$ and hence $b = e\,\varsigma^1 d$:

$$t((\lambda e)d) = (\lambda t(e))(t(d)) \to_{\beta} t(e)[\![\mathtt{1} \leftarrow t(d)]\!] \overset{L\,26}{=} t(e\,\varsigma^1 d)$$

$a = c\,\varsigma^i d$ **:** If the reduction takes place within c, i.e. $c \to_{\varsigma-gen} e$ and $b = e\,\varsigma^i d$, then

$$t(c\,\varsigma^i d) \overset{L\,26}{=} t(c)[\![\mathtt{i} \leftarrow t(d)]\!] \overset{IH\,\&\,L\,24.3}{\twoheadrightarrow_{\beta}} t(e)[\![\mathtt{i} \leftarrow t(d)]\!] \overset{L\,26}{=} t(e\,\varsigma^i d)$$

If the reduction takes place within $d$, lemma 24.2 applies.

$a = \theta_k c$ : The reduction must take place within $c$. Use lemma 26 and lemma 24.1. □

Now, the following corollaries are immediate.

**Corollary 2** *Let $a, b \in \Lambda t$ , if $a \twoheadrightarrow_{\lambda t} b$ then $t(a) \twoheadrightarrow_{\beta} t(b)$ .*

**Corollary 3 (Soundness)** *Let $a, b \in \Lambda$ , if $a \twoheadrightarrow_{\lambda t} b$ then $a \twoheadrightarrow_{\beta} b$ .*

This last corollary says that the $\lambda t$-calculus is correct with respect to the classical $\lambda$-calculus, i.e. derivations of pure terms ending with pure terms can also be derived in the classical $\lambda$-calculus.

Finally, before proving confluence, we verify that the $\lambda t$-calculus is powerful enough to simulate $\beta$-reduction.

**Lemma 28 (Simulation of $\beta$-reduction)** *Let $a, b \in \Lambda$, if $a \to_{\beta} b$ then $a \twoheadrightarrow_{\lambda t} b$ .*

**Proof:** Induction on $a$. As usual the interesting case is when $a = (\lambda c)d$ and $b = c[\![1 \leftarrow d]\!]$:

$$(\lambda c)d \to_{\varsigma-gen} c\varsigma^1 d \twoheadrightarrow_t t(c\varsigma^1 d) \overset{L26}{=} t(c)[\![1 \leftarrow t(d)]\!] \overset{c,d\in\Lambda}{=} c[\![1 \leftarrow d]\!]$$ □

**Theorem 8 (Confluence of $\lambda s$)** *The $\lambda t$-calculus is confluent on $\Lambda t$.*

**Proof:** We interpret the $\lambda t$-calculus into the $\lambda$-calculus via $t$-normalisation. We have:

$$\begin{tikzcd}
 & b \arrow[rr, two heads, "t"] & & t(b) \arrow[dr, two heads, "\beta"] & \\
a \arrow[ur, two heads, "\lambda t"] \arrow[rr, two heads, "t"] \arrow[dr, two heads, "\lambda t"'] & & t(a) \arrow[ur, two heads, "\beta"] \arrow[dr, two heads, "\beta"'] & \text{Thm. 4} & d \\
 & c \arrow[rr, two heads, "t"] & & t(c) \arrow[ur, two heads, "\beta"'] &
\end{tikzcd}$$

The existence of the arrows $t(a) \twoheadrightarrow_{\beta} t(b)$ and $t(a) \twoheadrightarrow_{\beta} t(c)$ is guaranteed by Corollary 2. We can close the diagram thanks to the confluence of the $\lambda$-calculus and finally lemma 28 ensures $t(b) \twoheadrightarrow_{\lambda t} d$ and $t(c) \twoheadrightarrow_{\lambda t} d$ proving thus CR for the $\lambda t$-calculus. □

## 5 Interpretation of $\lambda t$ into $\lambda_{\mathbf{exp}}$

The function that interprets $\lambda t$ into $\lambda_{\mathbf{exp}}$ is an extension of the function $u : \Lambda \to \Lambda_V$ (cf. Definition 18). Before introducing it, we must extend the notion of free variable.

**Definition 25** *The* set of free variables *of a term in $\Lambda t$ is defined by extending Definition 9 as follows:*

$$FV(\theta_k a) = FV(a)_{\leq k} \cup (FV(a)_{>k} + 1)$$
$$FV(a\varsigma^i b) = FV(a)_{<i} \cup (FV(a)_{>i} \setminus 1) \cup FV(b)$$

**Definition 26** *Let $a \in \Lambda t$ such that $FV(a) \subseteq \{1, \ldots, n\}$ and let $x_1, \ldots, x_n \in V$. We define $u_{[x_1,\ldots,x_n]}(d)$ by extending Definition 17 as follows:*

$$u_{[x_1,\ldots,x_n]}(a\,\varsigma^i b) = \begin{cases} u_{[x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(a)\sigma_x u_{[x_1,\ldots,x_n]}(b) & \text{if } n \geq i\,,\ x \notin \{x_1,\ldots,x_n\} \\ u_{[x_1,\ldots,x_n]}(a)\sigma_x u_{[x_1,\ldots,x_n]}(b) & \text{if } n < i\,,\ x \notin \{x_1,\ldots,x_n\} \end{cases}$$

$$u_{[x_1,\ldots,x_n]}(\theta_k a) = \begin{cases} u_{[x_1,\ldots,x_k,x_{k+2},\ldots,x_n]}(a) & \text{if } n > k+1 \\ u_{[x_1,\ldots,x_k]}(a) & \text{if } n = k+1 \\ u_{[x_1,\ldots,x_n]}(a) & \text{if } n < k+1 \end{cases}$$

In order to check that Definition 26 is correct, the following remark, whose proof is easy, is needed.

**Remark 4** *Let $a$, $b \in \Lambda t$.*

1. *If $FV(\lambda a) \subseteq \{1, \ldots, n\}$ then $FV(a) \subseteq \{1, \ldots, n+1\}$.*
2. *If $FV(a\,\varsigma^i b) \subseteq \{1, \ldots, n\}$ then $FV(b) \subseteq \{1, \ldots, n\}$ and*
   *if $n \geq i$ then $FV(a) \subseteq \{1, \ldots, n+1\}$ else $FV(a) \subseteq \{1, \ldots, n\}$.*
3. *If $FV(\theta_k a) \subseteq \{1, \ldots, n\}$ then*
   *if $n \geq k+1$ then $FV(a) \subseteq \{1, \ldots, n-1\}$ else $FV(a) \subseteq \{1, \ldots, n\}$.*

Furthermore, the definition of $u$ for abstractions and substitutions does not depend on the choice of the variable $x$ thanks to the following lemma.

**Lemma 29** *Let $a$, $b \in \Lambda t$ such that $FV(a) \subseteq \{1, \ldots, n+1\}$ and let $x_1, \ldots, x_n$ distinct variables and $x$, $y$ variables such that $x, y \notin \{x_1, \ldots, x_n\}$.*
*Then $\lambda x.u_{[x,x_1,\ldots,x_n]}(a) \equiv \lambda y.u_{[y,x_1,\ldots,x_n]}(a)$ and*
*$u_{[x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(a)\sigma_x b \equiv u_{[x_1,\ldots,x_{i-1},y,x_i,\ldots,x_n]}(a)\sigma_y b$.*

**Proof:** It is an immediate consequence of the following lemma. □

**Lemma 30** *Let $b \in \Lambda$ such that $FV(b) \subseteq \{1, \ldots, n+m+1\}$, and let the variables $x_1, \ldots, x_n$, $z_1, \ldots, z_m$, $x$ and $y$ be all distinct . Then $(u_{[z_1,\ldots,z_m,x,x_1,\ldots,x_n]}(b))[x := y] \equiv u_{[z_1,\ldots,z_m,y,x_1,\ldots,x_n]}(b)$.*

**Proof:** By induction on $b$. The two interesting cases are $b = \lambda a$ and $b = a\,\varsigma^i c$. Since the treatment of the second is analogous to the first one, we just study $b = \lambda a$.

Let us denote $\overline{x} = x_1, \ldots, x_n$ and $\overline{z} = z_1, \ldots, z_m$.

Let $u_{[\overline{z},x,\overline{x}]}(b) = \lambda w.u_{[w,\overline{z},x,\overline{x}]}(a)$. Let $u_{[\overline{z},y,\overline{x}]}(b) = \lambda v.u_{[v,\overline{z},y,\overline{x}]}(a)$.

Remark that we can assume that $w \neq y$. In fact, if $w = y$ we can choose $z$ such that $z \neq y$ and also distinct from $x_1, \ldots, x_n$, $z_1, \ldots, z_m$, $x$, and we have

$$u_{[\overline{z},x,\overline{x}]}(b) \equiv \lambda z.u_{[w,\overline{z},x,\overline{x}]}(a)[w := z] \overset{IH}{\equiv} \lambda z.u_{[z,\overline{z},x,\overline{x}]}(a)$$

Therefore, since $w \neq y$, we have

$$(u_{[\overline{z},x,\overline{x}]}(b))[x := y] = (\lambda w.u_{[w,\overline{z},x,\overline{x}]}(a))[x := y] = \lambda w.u_{[w,\overline{z},x,\overline{x}]}(a)[x := y] \overset{IH}{\equiv}$$

$$\lambda w.u_{[w,\overline{z},y,\overline{x}]}(a) \equiv \lambda v.u_{[w,\overline{z},y,\overline{x}]}(a)[w := v] \overset{IH}{\equiv} \lambda v.u_{[v,\overline{z},y,\overline{x}]}(a) = u_{[\overline{z},y,\overline{x}]}(b)$$ □

**Definition 27** *Let $\{v_1, \ldots, v_n, \ldots\}$ be the same enumeration of $V$ as in Definitions 16 and 18, we define $u : \Lambda t \to \Lambda\mathbf{exp}$ as the function given by $u(a) = u_{[v_1,\ldots,v_n]}(a)$ where $n$ is such that $FV(a) \subseteq \{1, \ldots, n\}$.*

The definition is correct thanks to the following lemma.

**Lemma 31** *If $a \in \Lambda t$, $FV(a) \subseteq \{1, \ldots, n\}$ and $m > n$ then $u_{[v_1,\ldots,v_n]}(a) = u_{[v_1,\ldots,v_m]}(a)$.*

**Proof:** Easy induction on $a$. $\square$

Remark that $u$ is not one-to-one. Indeed, $u$ cannot tell the difference between terms and their updatings, when they are $t$-equivalent. For instance, $u(\theta_0 \mathtt{1}) = v_1 = u(\mathtt{1})$.

**Lemma 32** *Let $a$, $b \in \Lambda t$, if $a \to_{\lambda t} b$ then $FV(b) \subseteq FV(a)$.*

**Proof:** By induction on $a$. If the reduction is internal the conclusion follows immediately from the IH. If the reduction is at the root, we must check that for every rule $a \to b$ we have $FV(b) \subseteq FV(a)$. This is easily done using Remark 2. $\square$

**Theorem 9** *Let $a$, $b \in \Lambda t$.*

1. *If $a \to_t b$ then $u(a) \overset{=}{\to}_{\mathbf{exp}} u(b)$.*
2. *If $a \twoheadrightarrow_t b$ then $u(a) \twoheadrightarrow_{\mathbf{exp}} u(b)$.*
3. *If $a \to_{\varsigma-gen} b$ then $u(a) \to_{\lambda\mathbf{exp}} u(b)$.*

**Proof:** To prove the first item we prove that if $a \to_t b$ and $FV(a) \subseteq \{1, \ldots, n\}$ then $u_{[x_1,\ldots,x_n]}(a) \overset{=}{\to}_{\mathbf{exp}} u_{[x_1,\ldots,x_n]}(b)$.

Remark first that Lemma 32 guarantees the correct definition of $u_{[x_1,\ldots,x_n]}(b)$.

The proof is by induction on $a$. If the reduction is internal, the IH is enough to settle the lemma. We must check now that for every rule $a \to_t b$ the lemma holds. As an example we study the rule $\varsigma$-$\lambda$-*transition*:

If, for instance, $n \geq i$ we have:

$$u_{[x_1,\ldots,x_n]}((\lambda a)\,\varsigma^i b) = u_{[x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(\lambda a)\sigma_x u_{[x_1,\ldots,x_n]}(b) =$$

$$(\lambda y.u_{[y,x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(a))\sigma_x u_{[x_1,\ldots,x_n]}(b) =$$

$$(\lambda y.u_{[y,x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(a))\sigma_x u_{[y,x_1,\ldots,x_n]}(\theta_0 b) \to$$

$$\lambda y.(u_{[y,x_1,\ldots,x_{i-1},x,x_i,\ldots,x_n]}(a)\sigma_x u_{[y,x_1,\ldots,x_n]}(\theta_0 b)) =$$

$$\lambda y.u_{[y,x_1,\ldots,x_n]}(a\varsigma^{i+1}(\theta_0 b)) = u_{[x_1,\ldots,x_n]}(\lambda(a\varsigma^{i+1}(\theta_0 b)))$$

It is this case that shows why the rule $\sigma$-$\lambda$-*transition* of the $\lambda s$-calculus had to be changed into the rule $\varsigma$-$\lambda$-*transition* of the $\lambda t$-calculus.

Remark also that the $\theta$-rules are the ones that leave the translations unchanged, i.e. if $a \to_{\theta-\mathrm{rule}} b$ then $u_{[x_1,\ldots,x_n]}(a) = u_{[x_1,\ldots,x_n]}(b)$.

The second item is easily obtained by proving that if $a \twoheadrightarrow_t b$ then $u_{[x_1,\ldots,x_n]}(a) \twoheadrightarrow_{\mathbf{exp}} u_{[x_1,\ldots,x_n]}(b)$ by induction on the length of the derivation using the first item.

For the third item, we prove that if $a \rightarrow_{\varsigma-gen} b$ then $u_{[x_1,\ldots,x_n]}(a) \rightarrow_{\lambda\mathbf{exp}} u_{[x_1,\ldots,x_n]}(b)$ by induction on $a$. The interesting case arises when the reduction takes place at the root:

If $n > 0$ we have:

$$u_{[x_1,\ldots,x_n]}((\lambda a)b) = (\lambda x.u_{[x,x_1,\ldots,x_n]}(a))u_{[x_1,\ldots,x_n]}(b) \rightarrow$$

$$u_{[x,x_1,\ldots,x_n]}(a)\sigma_x u_{[x_1,\ldots,x_n]}(b) = u_{[x_1,\ldots,x_n]}(a\varsigma^1 b)$$

If $n = 0$ we have:

$$u_{[]}((\lambda a)b) = (\lambda x.u_{[x]}(a))u_{[]}(b) \rightarrow u_{[x]}(a)\sigma_x u_{[]}(b) \overset{(1)}{=} u_{[]}(a)\sigma_x u_{[]}(b) = u_{[]}(a\varsigma^1 b)$$

where equality (1) holds because of Lemma 31 (with $n = 0$ and $m = 1$) and the fact that $FV(a) = \phi$ (since $FV(a\varsigma^1 b) = \phi$, Remark 4 yields $FV(a) = \phi$). □

## 6 $\lambda t$ preserves strong normalisation

Using Theorem 9 and the PSN of $\lambda\mathbf{exp}$, we can show the PSN of $\lambda t$. In order to do that we must use the fact that $u$, when restricted to pure terms, is an isomorphism. As a matter of fact, a weaker hypothesis than the existence of an isomorphism is enough, namely that $u$, when restricted to pure terms, admits a left inverse which preserves reduction. This was proved in subsection 2.4.

**Theorem 10 (PSN of $\lambda t$)** *Every $\lambda$-term which is strongly normalising in the $\lambda$-calculus à la de Bruijn is also strongly normalising in the $\lambda t$-calculus.*

**Proof:** Since $a \in \lambda$-SN, Theorem 2 and Lemma 17 guarantee that $u(a)$ is strongly normalising in the classical sense. The Preservation Theorem for $\lambda\mathbf{exp}$ (see Theorem 1) ensures $u(a) \in \lambda\mathbf{exp}$-SN.

If we assume $a \notin \lambda t$-SN, let

$$a \rightarrow_{\lambda t} a_1 \rightarrow_{\lambda t} \ldots \rightarrow_{\lambda t} a_n \rightarrow_{\lambda t} \ldots$$

be an infinite derivation. Since the $t$-calculus is SN (see Theorem 7), this derivation must contain an infinity of $\varsigma$-generations:

$$a \twoheadrightarrow_t a'_1 \rightarrow_{\varsigma-gen} a'_2 \twoheadrightarrow_t \ldots \twoheadrightarrow_t a'_{2n+1} \rightarrow_{\varsigma-gen} a'_{2n+2} \twoheadrightarrow_t \ldots$$

Now, by Theorem 9.2 and 9.3, we have:

$$u(a) \twoheadrightarrow_{\mathbf{exp}} u(a'_1) \rightarrow_{\lambda\mathbf{exp}} u(a'_2) \twoheadrightarrow_{\mathbf{exp}} \ldots \twoheadrightarrow_{\mathbf{exp}} u(a'_{2n+1}) \rightarrow_{\lambda\mathbf{exp}} u(a'_{2n+2}) \twoheadrightarrow_{\mathbf{exp}} \ldots$$

and this contradicts the fact that $u(a) \in \lambda\mathbf{exp}$-SN. Therefore, $a \in \lambda t$-SN. □

## 7 Comparison with $\lambda\sigma$ and $\lambda\upsilon$

For the syntax and rules of the $\lambda\sigma$ and $\lambda\upsilon$ calculi see [ACCL91] and [BBLRD95], respectively.

The $\lambda t$ calculus can be interpreted into the $\lambda\sigma$ calculus using a similar translation as the one presented in [KR95a] to interpret the $\lambda s$-calculus into $\lambda\sigma$. However, in the case of the $\lambda t$-calculus the interpretation works better: now $\lambda t$-derivations are preserved (only $s$-derivations and not $\lambda s$ derivations were preserved by the translation in [KR95a].)

In order to give the translation into the $\lambda\sigma$-calculus we give the following two definitions.

**Definition 28** *For $k \geq 0$ we define $s_k$ as follows: $s_0 = \uparrow$ and $s_k = \mathtt{1} \cdot \mathtt{2} \cdot \ldots \cdot \mathtt{k} \cdot \uparrow^{k+1}$.*

**Definition 29** *Let $b \in \Lambda\sigma^t$, we define a family of substitutions $(b_k)_{k\geq 1}$ as follows:*
$b_1 = b \cdot id \quad b_2 = \mathtt{1} \cdot b \cdot \uparrow \quad \ldots \quad b_{i+1} = \mathtt{1} \cdot \mathtt{2} \cdot \ldots \cdot \mathtt{i} \cdot b \cdot \uparrow^i \quad \ldots$

Using the rules *(Map)*, *(Clos)*, *(Ass)* and *(IdL)* it is easy to verify that:

**Remark 5** $\mathtt{1} \cdot (b_i \circ \uparrow) \twoheadrightarrow_\sigma (b[\uparrow])_{i+1}$ *and* $\mathtt{1} \cdot (s_k \circ \uparrow) \twoheadrightarrow_\sigma s_{k+1}$.

**Definition 30** *The* translation function $T : \Lambda s \to \Lambda\sigma^t$ *is defined by:*
$T(\mathtt{n}) = \mathtt{n} \quad T(a\,b) = T(a)T(b) \quad T(a\,\varsigma^i b) = T(a)[T(b)_i] \quad T(\lambda a) = \lambda(T(a)) \quad T(\theta_k a) = T(a)[s_k]$

**Theorem 11** *If $a \to_{\lambda t} b$ then $T(a) \xrightarrow{+}_{\lambda\sigma} T(b)$.*

**Proof:** Induction on $a$. We just check, as an example, the case $a = \mathtt{n}\,\varsigma^i c$ when the reduction takes place at the root:

$$T(\mathtt{n}\,\varsigma^i c) = \mathtt{n}[T(c)_i] \xrightarrow{+}_\sigma \begin{cases} \mathtt{n}-\mathtt{1} = T(\mathtt{n}-\mathtt{1}) & \text{if} \quad n > i \\ T(c) & \text{if} \quad n = i \\ \mathtt{n} = T(\mathtt{n}) & \text{if} \quad n < i \end{cases} \qquad \square$$

Even if $\lambda t$ is interpreted in $\lambda\sigma$ more faithfully than $\lambda s$ (the *$\sigma$-generation* rule translates (cf. [KR95a]) into a $\lambda\sigma$-equivalence rather than a derivation), no reasonable translation of $\lambda t$ into $\lambda\upsilon$ seems possible. The reason is that the operators of $\lambda\upsilon$ are not able to express, for instance, the $\lambda\sigma$-substitution $\mathtt{1}_2 = \mathtt{1} \cdot \mathtt{1} \cdot \uparrow$. Remark that in [KR95a] $\mathtt{1}_2$ was defined as $\mathtt{1} \cdot \mathtt{1}[\uparrow] \cdot \uparrow$, and this $\lambda\sigma$-substitution is avilable in the $\lambda\upsilon$ syntax as $\Uparrow(\mathtt{1}/)$.

The rest of this section will be devoted to compare the length of the derivations which simulate $\beta$-reduction in $\lambda t$ and $\lambda\upsilon$. We choose now $\lambda\upsilon$ instead of $\lambda\sigma$ because derivations are shorter in $\lambda\upsilon$ than in $\lambda\sigma$. We are going to prove that $\beta$-simulation in $\lambda t$ (one step *$\varsigma$-generation* followed by $t$-derivation to normal form) is more efficient than $\beta$-simulation in $\lambda\upsilon$ (one step $B$ followed by $\upsilon$-derivation to normal form).

We begin by introducing a set of terms $\Lambda_\theta$ on which induction will be used to define a function that computes the length of certain derivations. We are mainly interested in pure terms, which are contained in $\Lambda_\theta$, but the introduction of $\Lambda_\theta$ is necessary since it provides a strong induction hypothesis to prove the auxiliary results needed.

**Definition 31** $\Lambda_\theta ::= \mathbb{N} \mid \Lambda_\theta\Lambda_\theta \mid \lambda\Lambda_\theta \mid \theta_k\Lambda_\theta$ *, where $k \geq 0$. The* length *of terms in $\Lambda_\theta$ is defined by:*
$L_\theta(\mathtt{n}) = 1 \quad L_\theta(ab) = L_\theta(a) + L_\theta(b) + 1 \quad L_\theta(\lambda a) = L_\theta(\theta_k a) = L_\theta(a) + 1$ .
*By induction on $a \in \Lambda_\theta$ we mean induction on $L_\theta(a)$.*

**Remark 6** *Let $a \in \Lambda_\theta$ and $k \geq 0$, then $L_\theta(a) \geq L_\theta(t(\theta_k a))$.*

**Proof:** By induction on $a$. The interesting case is when $a = \theta_m b$. By IH we have $L_\theta(b) \geq L_\theta(t(\theta_m b))$ and since $L_\theta(a) > L_\theta(b)$, we apply again the IH (now to $L_\theta(t(\theta_m b))$) to obtain $L_\theta(t(\theta_m b)) \geq L_\theta(t(\theta_k(t(\theta_m b)))) = L_\theta(t(\theta_k(\theta_m b)))$. Hence, $L_\theta(a) \geq L_\theta(t(\theta_k a))$. □

The next remark will be used frequently without explicit mention.

**Remark 7** *If $a \in \Lambda_\theta$ and $a \to_t b$ then $b \in \Lambda_\theta$.*

**Proof:** Easy induction on $a$. □

**Definition 32** *We define $M : \Lambda_\theta \to \mathbb{N}$ by induction as follows:*
$M(\mathtt{n}) = 1 \quad M(ab) = M(a) + M(b) + 1 \quad M(\lambda a) = M(a) + 1 \quad M(\theta_k a) = M(t(\theta_k a)) + M(a)$

Remark that the definition is correct thanks to remark 6.

**Lemma 33** *For $a \in \Lambda_\theta$, every $t$-derivation of $\theta_k a$ to its $t$-normal form has length $M(a)$.*

**Proof:** By induction on the weight $W(a) = (W_1(a), W_2(a))$ used to prove SN for the $t$-calculus (see proof of Theorem 7). The basic case ($a = \mathtt{n}$) is immediate, since all the derivations of $\theta_k \mathtt{n}$ to its nf have length 1. We proceed now by a case analysis. We just treat the case $a = bc$ since the argument is similar for the other cases.

Let us consider a derivation $\mathcal{D}$ of $\theta_k(bc)$ to its nf.

If the first step is internal, say $b \to b'$, we know by IH ($P(b'c) < P(bc)$) that every derivation of $\theta_k(b'c)$ to its nf has length $M(b'c) = M(b') + M(c) + 1$. But IH (now applied to $b$ ($P(b) < P(bc)$) and $b'$ ($P(b') < P(bc)$) and the fact that $\theta_k b \to \theta_k b'$) also gives $M(b') = M(b) - 1$. Hence $M(b'c) = M(b) + M(c) = M(bc) - 1$. Therefore, the length of $\mathcal{D}$ is $M(bc)$.

If the first step is $\theta_k(bc) \to \theta_k(b)\theta_k(c)$, since there are no rules in $t$ which contract an application, every derivation of $\theta_k(b)\theta_k(c)$ to its nf, has length (IH applied to $b$ and $c$) $M(b) + M(c) = M(bc) - 1$. Therefore, the length of $\mathcal{D}$ is again $M(bc)$. □

**Corollary 4** *For $a \in \Lambda_\theta$, all the $t$-derivations of $\theta^i_k a$ to its $t$-normal form have the same length, namely $(i-1)M(t(a)) + M(a)$.*

**Proof:** Prove first by induction on $a \in \Lambda_\theta$ that $M(t(a)) = M(t(\theta_k a))$, then use Lemma 33 to prove the corollary. □

Now we are going to prove the corresponding results for $\lambda v$. Since the proofs are analogous, we just state the results.

**Definition 33** $\Lambda_\uparrow ::= \ \mathbb{N} \mid \Lambda_\uparrow\Lambda_\uparrow \mid \lambda\Lambda_\uparrow \mid \Lambda_\uparrow[\Uparrow^k(\uparrow)]$ *, where $k \geq 0$. The* length *of terms in $\Lambda_\uparrow$ is given by: $L_\uparrow(\mathtt{n}) = 1 \quad L_\uparrow(ab) = L_\uparrow(a) + L_\uparrow(b) + 1 \quad L_\uparrow(\lambda a) = L_\uparrow(a[\Uparrow^k(\uparrow)]) = L_\uparrow(a) + 1$ .*

**Remark 8** *Let $a \in \Lambda_\uparrow$ and $k \geq 0$, then $L_\uparrow(a) \geq L_\uparrow(v(a[\Uparrow^k(\uparrow)]))$.*

**Remark 9** *If $a \in \Lambda_\uparrow$ and $a \to_t b$ then $b \in \Lambda_\uparrow$.*

**Definition 34** *For $k \geq 0$, we define $M_k : \Lambda_\theta \to \mathbb{N}$ as follows:*

$$M_k(\mathtt{n}) = \begin{cases} 2k+1 & \text{if } n > k \\ 2n-1 & \text{if } n \leq k \end{cases} \qquad \begin{array}{ll} M_k(ab) = M_k(a) + M_k(b) + 1 & M_k(\lambda a) = M_k(a) + 1 \\ M_k(a[\Uparrow^p(\uparrow)]) = M_k(v(a[\Uparrow^p(\uparrow)])) + M_p(a) & \end{array}$$

**Lemma 34** *For $a \in \Lambda_{\uparrow}$, all the $v$-derivations of $a[\Uparrow^k(\uparrow)]$ to its $v$-nf have length $M_k(a)$.*

**Proof:** By induction on the weight used to show SN for the $v$-calculus (cf. [BBLRD95]) and case analysis. □

**Corollary 5** *For $a \in \Lambda_{\uparrow}$, all the $v$-derivations of $a[\Uparrow^k(\uparrow)]^i$ to its $v$-normal form have the same length, namely $(i-1)M_k(v(a)) + M_k(a)$.*

**Lemma 35** *Let $b \in \Lambda$, for every derivation $b[\Uparrow^k(\uparrow)]^i \twoheadrightarrow_v^m v(b[\Uparrow^k(\uparrow)]^i)$ there exists $n \le m$ such that $\theta_p^i b \twoheadrightarrow_t^n t(\theta_p^i b)$.*

**Proof:** Prove first that for every $b \in \Lambda$ and $k \ge 0$, $M_k(b) \ge M(b)$ by induction on $b \in \Lambda$. Conclude using lemmas 33 and 34. □

**Definition 35** *Let $a, b \in \Lambda$ and $i \ge 0$, we define $P_i(a,b)$ by induction on $a$:*

$$P_i(\mathtt{n}, b) = \begin{cases} 2i+1 & \text{if } \ n > i+1 \\ 2n-1 & \text{if } \ n < i+1 \\ i(1+M_0(b))+1 & \text{if } \ n = i+1 \end{cases} \qquad \begin{array}{l} P_i(cd,b) = P_i(c,b) + P_i(d,b) + 1 \\ P_i(\lambda c, b) = P_i(c,b) + 1 \end{array}$$

**Lemma 36** *Let $a, b \in \Lambda$ and $i \ge 0$, all the $v$-derivations of $a[\Uparrow^i(b/)]$ to its $v$-nf have the same length, namely $P_i(a,b)$.*

**Proof:** Easy induction on $a \in \Lambda$. □

**Lemma 37** *Let $a, b \in \Lambda$ and $i \ge 0$, there exists a derivation of $a\sigma^{i+1}(\theta_0^i b)$ to its $t$-nf whose length is less than or equal to $P_i(a,b)$.*

**Proof:** By induction on $a$ reducing always at the root. For the case $a = \mathtt{i+1}$ use the fact that $M_0(b) \ge M(b)$ (see proof of Lemma 35). □

**Theorem 12** *$\beta$-simulation is more efficient in $\lambda t$ than in $\lambda v$.*

**Proof:** We prove that for every $a \in \Lambda$ and every $\lambda v$-derivation $a \to_B b \twoheadrightarrow_v^m v(b)$ there exists $n \le m$ such that $a \to_{\sigma-gen} c \twoheadrightarrow_t^n t(c)$ by induction on $a$.

The interesting case is $a = (\lambda d)e \to_B d[e/] \twoheadrightarrow^m v(d[e/])$. By Lemma 36 we know that $m = P_0(d,e)$ and Lemma 37 gives a derivation $d\sigma^1 e \twoheadrightarrow_t^n t(d\sigma^1 e)$ such that $n \le P_0(d,e)$.

Remark that there are an infinity of cases for which the inequality is strict. For instance, let us consider the term $(\lambda\lambda\ldots\lambda.n)a$ with $m$ $\lambda$'s and $n > m > 1$. It is easy to check, using the function $P_{m-1}$ defined above that $3m-2$ reductions are needed to simulate $\beta$-reduction in $\lambda v$, whereas only $m+1$ reductions are sufficient in $\lambda t$. Remark that for $m > n$ the number of reductions needed in $\lambda v$ is also strictly greater than the number needed in $\lambda t$. □

## 8 About extensions on open terms

We end our work by pointing out the difficulties that arise when trying to extend $\lambda t$ to a confluent calculus on open terms.

Let us recall that such an extension was successful for $\lambda s$ and gave rise to the confluent calculus $\lambda s_e$ (cf. [KR96]).

**Definition 36** *The set of* open terms*, denoted* $\Lambda t_{op}$*, is given as follows:*

$$\Lambda t_{op} ::= \mathcal{V} \mid \mathbb{N} \mid \Lambda t_{op}\Lambda t_{op} \mid \lambda\Lambda t_{op} \mid \Lambda t_{op}\varsigma^i\Lambda t_{op} \mid \theta_k\Lambda t_{op} \qquad \textit{where} \quad i \geq 1\,, \quad k \geq 0$$

*and where* $\mathcal{V}$ *stands for a set of variables, over which* $X$*,* $Y$*, ... range. We take* $a$*,* $b$*,* $c$ *to range over* $\Lambda t_{op}$*. Furthermore,* closures*,* pure terms *and* compatibility *are defined as for* $\Lambda t$*.*

Working with open terms one loses confluence as shown by the following counterexample:

$$((\lambda X)Y)\varsigma^1\mathtt{1} \to (X\varsigma^1Y)\varsigma^1\mathtt{1} \qquad ((\lambda X)Y)\varsigma^1\mathtt{1} \to ((\lambda X)\varsigma^1\mathtt{1})(Y\varsigma^1\mathtt{1})$$

and $(X\varsigma^1Y)\varsigma^1\mathtt{1}$ and $((\lambda X)\varsigma^1\mathtt{1})(Y\varsigma^1\mathtt{1})$ have no common reduct. Moreover, the above example shows that even local confluence is lost.

When studying the same counterexample for $\lambda s$, we found that, since $((\lambda X)\sigma^1\mathtt{1})(Y\sigma^1\mathtt{1}) \to\!\!\!\to (X\sigma^2\mathtt{1})\sigma^1(Y\sigma^1\mathtt{1})$, the solution to the problem seemed at hand if one had in mind the properties of meta-substitutions and updating functions of the $\lambda$-calculus in the Bruijn notation (cf. lemmas 1 - 6). These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to $\lambda s$ give origin to a rewriting system which happens to be locally confluent (cf. [KR95b]). For instance, the rule corresponding to the Meta-substitution lemma (lemma 4) is the *$\sigma$-$\sigma$-transition* rule given below.

$$\sigma\text{-}\sigma\textit{-transition} \quad (a\,\sigma^i b)\,\sigma^j\,c \quad \longrightarrow \quad (a\,\sigma^{j+1}\,c)\;\sigma^i\,(b\,\sigma^{j-i+1}\,c) \quad \text{if} \quad i \leq j$$

The addition of this rule solves the critical pair for $\lambda s$, since now we have $(X\sigma^1Y)\sigma^1\mathtt{1} \to (X\sigma^2\mathtt{1})\sigma^1(Y\sigma^1\mathtt{1})$.

Following the same method we can try an orientation of the equality given in Lemma 23 to find our *$\varsigma$-$\varsigma$-transition* rule:

$$\varsigma\text{-}\varsigma\textit{-transition} \quad (a\,\varsigma^i b)\,\varsigma^j\,\theta_0^{i-1}c \quad \longrightarrow \quad (a\,\varsigma^{j+1}\,\theta_0^i c)\;\varsigma^i\,(b\,\varsigma^j\,\theta_0^{i-1}c) \quad \text{if} \quad i \leq j$$

Remark that in the *$\sigma$-$\sigma$-transition* rule no such operator appears. This new situation gives rise to undesirable critical pairs. For instance:

$$(a\,\varsigma^i b)\,\varsigma^j\,\theta_0^{i-1}(\lambda d) \to (a\,\varsigma^{j+1}\,\theta_0^i(\lambda d))\;\varsigma^i\,(b\,\varsigma^j\,\theta_0^{i-1}(\lambda d))$$
$$(a\,\varsigma^i b)\,\varsigma^j\,\theta_0^{i-1}(\lambda d) \to (a\,\varsigma^i b)\,\varsigma^j\,\lambda(\theta_1^{i-1}c)$$

Since these critical pairs cannot be solved without creating new ones, we can try another approach to our problem: consider a generalization of the *$\varsigma$-$\varsigma$-transition* rule that avoids the occurrence of the $\theta$ operator in the left hand side:

$$\textit{new } \varsigma\text{-}\varsigma\textit{-transition} \quad (a\,\varsigma^i b)\,\varsigma^j\,c \quad \longrightarrow \quad (a\,\varsigma^{j+1}\,\theta_0 c)\;\varsigma^i\,(b\,\varsigma^{j-i+1}\,c) \quad \text{if} \quad i \leq j$$

But this rule is not correct. Indeed, it is easy to check that with it, it is possible to derive $(\mathtt{3}\varsigma^2\mathtt{3})\varsigma^2\mathtt{1} \twoheadrightarrow \mathtt{2}$ while if only *$\varsigma$-destruction* is used the derivation is $(\mathtt{3}\varsigma^2\mathtt{3})\varsigma^2\mathtt{1} \twoheadrightarrow \mathtt{1}$.

Therefore, the $\lambda t$-calculus does not seem to possess a reasonable extension on open terms.

## 9 Conclusion

Even if the $\lambda t$-calculus cannot be extended to a confluent extension on open terms (of the calculi mentioned in the Introduction, only the $\lambda s$-calculus, the $\lambda\sigma_{\Uparrow}$-calculus of the $\lambda\sigma$-family and the $\lambda\zeta$-calculus enjoy this property; furthermore, $\lambda\sigma_{\Uparrow}$ and $\lambda\zeta$ are themselves confluent on open terms), it happens to be an interesting calculus for two reasons:

1. It can be related to $\lambda\mathbf{exp}$, as we have shown in this paper, via an immersion which is an extension of the classical isomorphism between the classical $\lambda$-caculus and its de Bruijn version.
2. While being a calculus à la $\lambda s$, it works with partial updatings and this is a feature that characterizes the $\lambda\sigma$-calculi, the $\lambda\upsilon$-calculus and the $\lambda\zeta$-calculus. Therefore, it offers a new perspective between the $\lambda s$ and the $\lambda\sigma$ styles.
3. It simulates $\beta$-reduction more efficiently than $\lambda\upsilon$ which seems to be the most efficient of the calculi in the $\lambda\sigma$-style.

One of the questions we raised in the Introduction is still unanswered, namely if the $\lambda\mathbf{exp}$-calculus is isomorphic to a calculus in de Bruijn notation which could be described in a satisfactory manner. Our attempts to show that there is an inmersion in the other direction have failed and we conclude this paper by pointing out the problems that arise when trying to define such an immersion, i.e. the inmersion of $\lambda\mathbf{exp}$ into $\lambda t$.

Now the question is how to extend the functions $w_{[x_1,\ldots,x_n]}$ given in Definition 15. Therefore we must define $w_{[x_1,\ldots,x_n]}(a\sigma_x b)$. Since

$$w_{[x_1,\ldots,x_n]}((\lambda x.a)b) = (\lambda w_{[x,x_1,\ldots,x_n]}(a))(w_{[x_1,\ldots,x_n]}(b)) \to w_{[x,x_1,\ldots,x_n]}(a)\varsigma^1 w_{[x_1,\ldots,x_n]}(b)$$

and since we want the $w_{[x_1,\ldots,x_n]}$'s to preserve reduction we are tempted to define

$$w_{[x_1,\ldots,x_n]}(a\sigma_x b) = w_{[x,x_1,\ldots,x_n]}(a)\varsigma^1 w_{[x_1,\ldots,x_n]}(b)$$

But this definition is not good enough to preserve other rules, for instance

$$w_{[x_1,\ldots,x_n]}((\lambda x.a)\sigma_y b) = (\lambda w_{[x,y,x_1,\ldots,x_n]}(a))\varsigma^1 w_{[x_1,\ldots,x_n]}(b)$$

$$\to \lambda(w_{[x,y,x_1,\ldots,x_n]}(a)\varsigma^2\theta_0(w_{[x_1,\ldots,x_n]}(b)))$$

whereas

$$w_{[x_1,\ldots,x_n]}(\lambda x.(a\sigma_y b)) = \lambda(w_{[y,x,x_1,\ldots,x_n]}(a)\varsigma^2 w_{[x,x_1,\ldots,x_n]}(b))$$

and we see that the variables $y$ and $x$ are now in inverted positions.

We realize that our $w_{[x_1,\ldots,x_n]}$'s should "know" how many $\lambda$'s have been crossed and act accordingly, i.e. placing the variable of the substitution in the right place. In order to achieve this we should introduce families of translations $w^i_{[x_1,\ldots,x_n]}$, with $i \geq 0$, and the translation we are trying to define should be $w^0_{[x_1,\ldots,x_n]}$. Therefore we propose to define (we restrict the definition to abstraction and substitution since the difficulty already appears with these rules):

$$\begin{array}{l} w^i_{[x_1,\ldots,x_n]}(\lambda x.a) = \lambda w^{i+1}_{[x,x_1,\ldots,x_n]}(a) \\ w^i_{[x_1,\ldots,x_n]}(a\sigma_x b) = w^{i+1}_{[x_1,\ldots,x_k,x,x_{k+1},\ldots,x_n]}(a)\varsigma^{k+1} w^i_{[x_1,\ldots,x_n]}(b) \end{array}$$

The reader can easily check that with this definition reduction is now preserved for the $\sigma$-$\lambda$-*transition* rule of the $\lambda_{\mathbf{exp}}$-calculus (assuming that a lemma analogous to Lemma 13 will hold for the operators $\theta_k$). But unfortunately the $\sigma$-*generation* rule is the one that fails now.

Therefore the question of the existence of an extension of $w$ preserving reduction remains still open. Furthermore, it is not clear what calculus of explicit substitutions à la de Bruijn could be isomorphic to $\lambda_{\mathbf{exp}}$. It may be that we have to go the other way round: find a calculus of explicit substitutions using variable names which could be proved isomorphic to one in de Bruijn notation. This is under investigation.

## References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Bar84] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. North Holland, 1984.

[BBLRD95] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Personal communication*, 1995.

[Blo95] R. Bloo. Preservation of Strong Normalisation for Explicit Substitution . Technical Report 95-08, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1995.

[CHL92] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report RR 1617, INRIA, Rocquencourt, 1992.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986. Revised edition : Birkhäuser (1993).

[dB72] N. de Bruijn. Lambda-Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[dB78a] N. de Bruijn. Lambda-Calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, 40:348–356, 1978.

[dB78b] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical Report TH-Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978.

[Har89] T. Hardin. Confluence Results for the Pure Strong Categorical Logic CCL : $\lambda$-calculi as Subsystems of CCL. *Theoretical Computer Science*, 65(2):291–342, 1989.

[Hue80] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the Association for Computing Machinery*, 27:797–821, October 1980.

[KB70] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KN93] F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.

[KN96] F. Kamareddine and R. P. Nederpelt. A useful $\lambda$-notation. *Theoretical Computer Science*, 155:85–109, 1996.

[KR95a] F. Kamareddine and A. Ríos. A $\lambda$-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *Lecture Notes in Computer Science*, 982:45–62, 1995.

[KR95b] F. Kamareddine and A. Ríos. The $\lambda s$-calculus: its typed and its extended versions. Technical report, Department of Computing Science, University of Glasgow, 1995.

[KR96] F. Kamareddine and A. Ríos. The confluence of the $\lambda s_e$-calculus via a generalized interpretation method. Technical report, Glasgow University, 1996.

[Mel95] P.-A. Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate in Proceedings of TLCA'95. *Lecture Notes in Computer Science*, 902, 1995.

[MH95] C. A. Muñoz Hurtado. Confluence and preservation of strong normalisation in an explicit substitutions calculus. Technical report, INRIA, Rocquencourt, 1995. 2762.

[Río93] A. Ríos. *Contribution à l'étude des $\lambda$-calculs avec substitutions explicites.* PhD thesis, Université de Paris 7, 1993.