Bridging de Bruijn Indices and Variable Names in Explicit Substitutions Calculi

FAIROUZ KAMAREDDINE, *Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland. E-mail: fairouz@dcs.gla.ac.uk*

ALEJANDRO RÍOS, *University of Buenos Aires, Cordoba 3571 11 A, 1188 Ciudad de Buenos Aires, Argentina. E-mail: arios@cactus.fi.uba.ar*

Abstract

Calculi of explicit substitutions have almost always been presented using de Bruijn indices with the aim of avoiding α-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. We believe that the study of these systems 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 and to do so, we provide the λt-calculus: a λ-calculus à la de Bruijn which can be translated into a λ-calculus with explicit substitutions written with variables names. We present explicitly this translation and use it to obtain preservation of strong normalisation for λt. Moreover, we show several properties of λt, including confluence on closed terms and efficiency to simulate β-reduction. Furthermore, λt is a good example of a calculus written in the λs-style (cf. [19]) that possesses the updating mechanism of the calculi à la $\lambda\sigma$ (cf. [1, 7, 26]).

Keywords: lambda calculus, variable names, de Bruijn indices, explicit substitutions

1 Introduction

The classical λ-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 λ-calculus. There has however been an interest in formalising substitution explicitly in order to provide a theoretical framework for the implementation of functional programming languages. 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. [10]); the calculi of categorical combinators (cf. [8]); $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\sigma_{SP}$ (cf. [1, 7, 26]) referred to as the $\lambda\sigma$-family; $\varphi\sigma BLT$ (cf. [18]); $\lambda\upsilon$ (cf. [3]) and $\lambda\zeta$ (cf. [25]) which are descendants of the $\lambda\sigma$-family; λs (cf. [19]) and λs_e (cf. [22]).

All the calculi above mentioned are described in de Bruijn notation (cf. [9, 11]). 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 α-congruence, it provides term rewriting systems instead of just abstract rewriting systems and therefore more rewriting tools are available to study them. The only 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, $\lambda\mathsf{x}$, has been introduced and studied(cf. [27, 5, 6, 28, 4]). 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 λ-calculus is also strongly normalising in the $\lambda\mathsf{x}$-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 [24]. As a matter of fact, of the above mentioned calculi only $\lambda\upsilon$, λs and $\lambda\zeta$ have PSN.

The following question poses itself: *is the $\lambda\mathsf{x}$-calculus equivalent to one of the already known calculi in de Bruijn notation (say λ_{sub})*[1]*, and, if not, can we describe $\lambda\mathsf{x}$ in de Bruijn notation in a satisfactory manner?* Trying to answer this question we realized that λs, which intuitively[2] was the best candidate for a de Bruijn version of $\lambda\mathsf{x}$, was not the answer. Thus we were led to a new calculus, which we call λt, whose formulation is slightly different from the formulation of λs and whose relationship with $\lambda\mathsf{x}$ can be, partly, explained. In particular, we find a function u that interprets λt into $\lambda\mathsf{x}$ preserving reductions. We point out the difficulty of reversing this process in order to find an interpretation from $\lambda\mathsf{x}$ into λt that preserves reductions and that is the inverse of u.

Although the rules of λt and λs are similar, both calculi work quite differently: while λs makes global updating when performing a substitution, the λt-calculus makes partial updating 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 λt-calculus can be considered as a calculus written in the λs-style which works with the updating mechanism of the $\lambda\sigma$-calculi. The plan of this paper is as follows:

- In Section 2 we introduce the formal machinery and we present the λ-calculus with variable names $(\Lambda_V, \rightarrow_\lambda)$ and the λ-calculus with de Bruijn indices $(\Lambda, \rightarrow_\beta)$. We define the notion of equivalence (or isomorphism) between two reduction systems and we show that $(\Lambda_V, \rightarrow_\lambda)$ and $(\Lambda, \rightarrow_\beta)$ are isomorphic in our sense. Finally, we present the λs-, $\lambda\sigma$- and $\lambda\upsilon$-calculi and note that the $\lambda\sigma$-calculus results from turning the meta-notions (meta-substitutions and meta-updating) of $(\Lambda, \rightarrow_\beta)$ into object notions.
- In Section 3, we present another formulation of the λ-calculus à la de Bruijn where the meta-notions work in a different manner to those of $(\Lambda, \rightarrow_\beta)$.
- In Section 4, we construct the λt-calculus from the new formulation (of Section 2)

[1]We take here equivalence to mean that there is a translation T_1 from $\lambda\mathbf{x}$ to λ_{sub} and another translation T_2 from λ_{sub} to $\lambda\mathbf{x}$ such that T_1 and T_2 are inverses of one another and they both preserve the corresponding reductions.

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

of the λ-calculus à la de Bruijn and we show that the t-calculus is strongly normalising and that the λt-calculus simulates β-reduction and is confluent (on closed terms) using the "interpretation method" ([14, 7]).

- In Section 5, we make explicit the relationship between λt and $\lambda\mathtt{x}$, which happens to be a sort of immersion, and we use this immersion to prove the PSN for λt using the PSN of $\lambda\mathtt{x}$.
- In Section 6, we compare λt with $\lambda\sigma$ by providing an interpretation of the former into the latter and argue about the impossibility of such an interpretation into $\lambda\upsilon$. We also prove that λt is more efficient (there exist shorter reduction paths) to simulate β-reduction than $\lambda\upsilon$.[3]
- In Section7, we discuss the problem of extending λt 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 a translation of $\lambda\mathtt{x}$ into λt.

2 Preliminaries

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

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 2.1 *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 $a \to_R b$ or $a \to b$ when the context is clear enough. We call* reduction *this relation and* reduction system*, the pair (A,R). We denote with $\overset{=}{\to}_R$ the reflexive closure of R, with $\twoheadrightarrow_R$ or just $\twoheadrightarrow$ the reflexive and transitive closure of R and with $\twoheadrightarrow_R^+$ or just $\twoheadrightarrow^+$ the 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 2.2 *Let R be a reduction on A.*

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

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

[3] $\lambda\upsilon$ may appear to be more efficient than $\lambda\sigma$. For example, $(\lambda 1(\lambda 23))3 \to_\beta 3\lambda(3[\uparrow])$ can be simulated more efficiently in $\lambda\upsilon$ than in $\lambda\sigma$. However, it is not always the case that simulation of β-reduction can be done in shorter steps in $\lambda\upsilon$ than in $\lambda\sigma$. For example, for $n \geq 2$, the reduction $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ can be simulated in $\lambda\sigma$ in $n+9$ steps whereas in $\lambda\upsilon$, it can be simulated in $4n+5$ steps. Obviously, $n+9 < 4n$ for $n \geq 2$. For more details, see [21].

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 2.3 *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 $a \to b$ and we say that b* has a normal form *if there exists a nf 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 $a_i \to a_{i+1}$ for all $i \geq 0$.*

Remark 2.4 *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 λ-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 λ which binds a variable represented by the number `n` you must travel upwards in the tree associated with the term and choose the n^{th} λ you find. For instance, $\lambda x.\lambda y.xy$ is written using de Bruijn indices as $\lambda\lambda(\mathtt{21})$ and $\lambda x.\lambda y.(x(\lambda z.zx))y$ is written as $\lambda\lambda(\mathtt{2}(\lambda(\mathtt{13}))\mathtt{1})$. 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 $\lambda\mathtt{34}$ whereas $\lambda x.zy$ translates as $\lambda\mathtt{43}$. The translations between both (classical and de Bruijn) notations will be given explicitly in Section 2.4.

The interest in introducing de Bruijn indices is that they avoid clashes of variables and therefore neither α-conversion nor Barendregt's convention are needed. Here is the λ-calculus à la de Bruijn.

Definition 2.5 *We define Λ, 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 Λ and $i, j, m, n, \ldots$ 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 Λ *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 β-reduction à la de Bruijn, we must define the substitution of a variable `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 `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 λ disappeared and to increment the free variables of $\lambda\mathtt{31}$ as they occur within the scope of one more λ.

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

Definition 2.6 *The* updating functions $U_k^i : \Lambda \to \Lambda$ *for* $k \geq 0$ *and* $i \geq 1$ *are defined inductively as follows:*

$$U_k^i(ab) = U_k^i(a)\, U_k^i(b) \qquad U_k^i(\mathtt{n}) = \begin{cases} \mathtt{n+i-1} & \textit{if} \;\; n > k \\ \mathtt{n} & \textit{if} \;\; n \leq k\,. \end{cases}$$
$$U_k^i(\lambda a) = \lambda(U_{k+1}^i(a))$$

The intuition behind U_k^i 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 2.7 *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:*

$$\begin{aligned}(a_1a_2)\{\!\{\mathtt{i} \leftarrow b\}\!\} &= (a_1\{\!\{\mathtt{i} \leftarrow b\}\!\})\,(a_2\{\!\{\mathtt{i} \leftarrow b\}\!\}) \\ (\lambda a)\{\!\{\mathtt{i} \leftarrow b\}\!\} &= \lambda(a\{\!\{\mathtt{i+1} \leftarrow b\}\!\})\end{aligned} \qquad \mathtt{n}\{\!\{\mathtt{i} \leftarrow b\}\!\} = \begin{cases} \mathtt{n-1} & \textit{if} \;\; n > i \\ U_0^i(b) & \textit{if} \;\; n = i \\ \mathtt{n} & \textit{if} \;\; n < i \end{cases}$$

Ultimately, the intention is to define $(\lambda a)b \to_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$ (see Definition 2.14 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 $n > i$) because one λ 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 that we have $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \to_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$, 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 λs. The proofs of Lemmas 2.8 - 2.13 are obtained by induction on a. Furthermore, the proof of Lemma 2.10 requires Lemma 2.9 with $p = 0$; the proof of Lemma 2.11 uses Lemmas 2.8 and 2.10 both with $k = 0$; finally, Lemma 2.12 with $p = 0$ is needed to prove Lemma 2.13.

Lemma 2.8 *For* $k < n \leq k+i$ *we have:* $U_k^i(a) = U_k^{i+1}(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$.

Lemma 2.9 *For* $p \leq k < j+p$ *we have:* $U_k^i(U_p^j(a)) = U_p^{j+i-1}(a)$.

Lemma 2.10 *For* $k+i \leq n$ *we have:* $U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n-i+1} \leftarrow b\}\!\})$.

Lemma 2.11 (Meta-substitution lemma) *For* $1 \leq i \leq n$ *we have:*
$$a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n+1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n-i+1} \leftarrow c\}\!\}\}\!\}$$

Lemma 2.12 *For* $p+j \leq k+1$ *we have:* $U_k^i(U_p^j(a)) = U_p^j(U_{k+1-j}^i(a))$.

Lemma 2.13 (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 2.14 *The* λ-calculus à la de Bruijn *is the reduction system* $(\Lambda, \rightarrow_\beta)$ *where* $\rightarrow_\beta$ *is the least compatible reduction on* Λ *generated by the single rule:*

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

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

Lemma 2.15 *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 $c = c_1 c_2$ and the reduction takes place at the root, i.e. $c_1 = (\lambda a)$, $c_2 = b$ and $d = a\{\!\{\mathtt{1} \leftarrow b\}\!\}$:

$$U^i_k((\lambda a)b) = (\lambda(U^i_{k+1}(a)))U^i_k(b) \rightarrow_\beta U^i_{k+1}(a)\{\!\{\mathtt{1} \leftarrow U^i_k(b)\}\!\} \overset{L\ 2.13}{=} U^i_k(a\{\!\{\mathtt{1} \leftarrow b\}\!\})$$

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 2.11 is required. ∎

This lemma was used in [19] to prove the confluence of λs. 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 2.16 *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 2.17 *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 Λ.

Definition 2.18 *The* set of free variables *of a term in Λ 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 2.19 *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 2.17.1 for the case $a = b\,c$ and Remark 2.17.2 for the case $a = \lambda b$. ∎

Lemma 2.20 *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 2.17.4, 5, 6 for the case $a = \lambda b$. ∎

Lemma 2.21 *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 λ-calculus and the $\lambda\mathtt{x}$-calculus

We assume the reader familiar with the λ-calculus (cf. [2]) in classical notation. We just recall the syntax of its terms and the definition of β-reduction.

Definition 2.22 *Given a set of variables $V = \{v_n : n \in \mathbb{N}\}$ we define recursively the set of terms Λ_V as follows:*

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

We use x, y, ... (with or without subscripts) to range over V and A, B, ... to range over λ_V. We assume that different variable names stand for different variables.

We say that a reduction $\to$ is compatible *on Λ_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 $\lambda x.A \to \lambda x.B$.*

α-congruent terms (terms which only differ on the name of bound variables) are identified and we use Barendregt's variable convention (all the bound variables of all the terms occurring in a certain mathematical context are chosen to be different from the free variables and from the other bound variables), abbreviated VC. Despite the use of VC, we shall sometimes stress the conditions on variables.

The classical notions of meta-substitution and α-congruence are defined as usual (cf. [2]). The meta-substitution of B for x in A is denoted by $A[x := B]$ and $A \equiv B$ means that A and B are α-congruent.

Definition 2.23 *The* λ-calculus *is the reduction system $(\Lambda_V, \to_\lambda)$, where $\to_\lambda$ is the least compatible reduction on Λ_V generated by:*

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

The $\lambda\mathbf{x}$-calculus of [27] is a calculus of explicit substitutions where variable names are used instead of de Bruijn numbers. Its set of rules is minimal and they mimick the definition of the meta-substitution by orientating its equalities. We will present below $\lambda\mathbf{x}$.

We begin by giving the syntax of the terms:

Definition 2.24 *Given a set of variables $V = \{v_n : n \in \mathbb{N}\}$ we define recursively the set of terms $\Lambda\mathbf{x}$ as follows:*

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

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

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

Definition 2.25 *The* set of free variables *of a term A, denoted $FV(A)$, the* meta-substitution *of B for x in a term A, denoted $A[x := B]$, and the notion of* α-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. $C\sigma_x D \equiv C[x := y]\sigma_y D$

α-congruent terms are identified and we assume again Barendregt's convention, but we have to extend it to deal with the new σs. Variables now are bound by both σ's and λ's and we assume that all the bound variables (by a λ or a σ) of all the terms occurring in a certain mathematical context are chosen to be different from the free variables and from the other bound variables.

Remark that, in order to make the notation lighter, we keep the same notation for the original notions and the extended ones. The context will be always clear enough to avoid confusions.

Definition 2.26 *The* $\lambda\mathbf{x}$-calculus *is the reduction system $(\Lambda\mathbf{x}, \rightarrow_{\lambda\mathbf{x}})$, where $\rightarrow_{\lambda\mathbf{x}}$ is the least compatible reduction on $\Lambda\mathbf{x}$ 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	*if $y \neq x$*

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

We use $\lambda\mathbf{x}$ to denote this set of rules. The calculus of substitutions associated with the $\lambda\mathbf{x}$-calculus is the rewriting system whose rules are $\lambda\mathbf{x} - \{\sigma\text{-generation}\}$ and we call it $\mathbf{x}$-calculus (in [5] it is called σ^-).

The main result in [5] is the preservation of strong normalisation of the $\lambda\mathbf{x}$-calculus with respect to classical λ-calculus:

Theorem 2.27 (PSN of $\lambda\mathbf{x}$) *Every term which is strongly normalising in the classical λ-calculus is also strongly normalising in the $\lambda\mathbf{x}$-calculus.*

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

It is well known that the classical λ-calculus and its de Bruijn version are isomorphic rewriting systems. Nevertheless we explicit here the isomorphism, since we are going to extend it later, and give the main lines of the proofs of the results we shall need.

We start by defining what we mean by isomorphism between two reduction systems and then we show that $(\Lambda_V, \rightarrow_\lambda)$ and $(\Lambda, \rightarrow_\beta)$ are isomorphic in our sense.

Definition 2.28 *We say that two reduction systems (A_1, R_1) and (A_2, R_2) are isomorphic if there exists two functions $w : A_1 \rightarrow A_2$ and $u : A_2 \rightarrow A_1$ such that:*

1. $\forall a_1, b_1 \in A_1, a_1 \rightarrow_{R_1} b_1 \Rightarrow w(a_1) \rightarrow_{R_2} w(b_1)$.
2. $\forall a_2, b_2 \in A_2, a_2 \rightarrow_{R_2} b_2 \Rightarrow u(a_2) \rightarrow_{R_1} u(b_2)$.
3. $w \circ u = u \circ w = Id$ *where Id is the identity function.*

As we need to use the set V in some order, we fix an enumeration of it:

Definition 2.29 *Throughout the paper, we take $\{v_1, \ldots, v_n, \ldots\}$ to be an enumeration of V.*

Definition 2.30 *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 α-congruent terms have the same image. This is a consequence of the following lemma.

Lemma 2.31 *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. ∎

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

Definition 2.32 *We define $w : \Lambda_V \to \Lambda$ as the function given by $w(A) = w_{[v_1,\ldots,v_n]}(A)$ where n is such that $FV(A) \subseteq \{v_1, \ldots, v_n\}$.*

The definition is correct in the following sense.

Lemma 2.33 *Let $A \in \Lambda_V$ such that $FV(A) \subseteq \{x_1, \ldots, x_n\}$ and let $y_1, \ldots, y_m$ be arbitrary variables. Then $w_{[x_1,\ldots,x_n,y_1,\ldots,y_m]}(A) = w_{[x_1,\ldots,x_n]}(A)$.*

Proof. Easy induction on A. ∎

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

Lemma 2.34 *Let $A \in \Lambda_V$, $k \geq 0$, $i \geq 1$ and $n \geq k+i$ such that $FV(A) \subseteq \{x_1, \ldots, x_n\} \setminus \{x_{k+1}, \ldots, x_{k+i-1}\}$. Then $w_{[x_1,\ldots,x_n]}(A) = U^i_k(w_{[x_1,\ldots,x_k,x_{k+i},\ldots,x_n]}(A))$.*

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

1. $A = v_m$ **:** Let $j = \min\{i : v_m = x_i\}$. Then $w_{[x_1,\ldots,x_n]}(v_m) = j$. If:
$j \leq k$ we have $w_{[x_1,\ldots,x_n]}(A) = \mathtt{j} = U^i_k(\mathtt{j}) = U^i_k(w_{[x_1,\ldots,x_k,x_{k+i},\ldots,x_n]}(A))$.
$j \geq k+i$ we have $w_{[x_1,\ldots,x_n]}(A) = \mathtt{j} = U^i_k(\mathtt{j} - \mathtt{i} + \mathtt{1}) = U^i_k(w_{[x_1,\ldots,x_k,x_{k+i},\ldots,x_n]}(A))$.

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

∎

Lemma 2.35 *Let A, $B \in \Lambda_V$ such that the bound variables of B are not free in A and let $i \geq 1$, $y_1, \ldots, y_{i-1} \notin FV(A)$, $\overline{y} = y_1, \ldots, y_{i-1}$, x be distinct from $y_1, \ldots, y_{i-1}$ and x not bound in B and take $\overline{x} = x_1, \ldots, x_n$.*
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:

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

$$w_{[\overline{y},\overline{x}]}(B[x := A]) = w_{[\overline{y},\overline{x}]}(A) \overset{L\,2.34}{=} U^i_0(w_{[\overline{x}]}(A)) = (w_{[\overline{y},x,\overline{x}]}(B))\{\!\{\mathtt{i} \leftarrow w_{[\overline{x}]}(A)\}\!\}$$

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

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

2. $B = \lambda z.D$ **:** 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 2.36 *Let A, $B \in \Lambda_V$, if $A \rightarrow_\lambda B$ then $w(A) \rightarrow_\beta w(B)$.*

PROOF. It is enough to show that if $FV(A) \subseteq \{x_1, \dots, x_n\}$ then $w_{[x_1,\dots,x_n]}(A) \rightarrow_\beta w_{[x_1,\dots,x_n]}(B)$.

Remark that since $FV(B) \subseteq FV(A)$ (cf. [2]), $w_{[x_1,\dots,x_n]}(B)$ 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

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

$$(w_{[x,x_1,\dots,x_n]}(D))\{\!\{\mathtt{1} \leftarrow w_{[x_1,\dots,x_n]}(E)\}\!\} \overset{L\,2.35}{=} w_{[x_1,\dots,x_n]}(D[x := E]) = w_{[x_1,\dots,x_n]}(B)$$

Remark that the conditions on the variables of Lemma 2.35 hold thanks to VC. ■

We give now the inverse of w:

Definition 2.37 *Let $a \in \Lambda$ such that $FV(a) \subseteq \{1, \dots, n\}$ and let $x_1, \dots, x_n \in V$. We define $u_{[x_1,\dots,x_n]}(a)$ by induction on a as follows:*

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

In order to check that Definition 2.37 is correct, we must verify that $FV(a) \subseteq \{1, \dots, n+1\}$ whenever $FV(\lambda a) \subseteq \{1, \dots, n\}$, which is obvious, and also that the definition of $u_{[x_1,\dots,x_n]}$ on abstractions does not depend on the choice of the variable x. This proof is analogous to the proof of Lemma 5.4 and Lemma 5.5, 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 Λ_V which depends on n where n is such that $FV(a) \subseteq \{1, \dots, n\}$. We remove now this condition and define a uniform translation on Λ.

Definition 2.38 *We define $u : \Lambda \rightarrow \Lambda_V$ as the function given by $u(a) = u_{[v_1,\dots,v_n]}(a)$ where n is such that $FV(a) \subseteq \{1, \dots, n\}$.*

The definition is correct thanks to Lemma 5.7 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 2.39 *Let $a \in \Lambda$, $i \geq 1$, $k \geq 0$ and $n \geq k + i$ such that $FV(a) \subseteq \{1, \dots, n - i + 1\}$. Then $u_{[x_1,\dots,x_n]}(U_k^i(a)) \equiv u_{[x_1,\dots,x_k,x_{k+i},\dots,x_n]}(a)$.*

PROOF. By induction on a. As usual we study the two interesting cases:

1. $a = \mathtt{m}$ **:** If $m \leq k$ then $u_{[x_1,\dots,x_n]}(U_k^i(a)) = x_m = u_{[x_n,\dots,x_{k+i},x_k,\dots,x_1]}(a)$.
If $m > k$ then $u_{[x_1,\dots,x_n]}(U_k^i(a)) = x_{m+i-1} = u_{[x_n,\dots,x_{k+i},x_k,\dots,x_1]}(a)$.

2. $a = \lambda b$ **:** We can choose x to obtain:

$$u_{[x_1,\ldots,x_n]}(U^i_k(a)) = \lambda x.u_{[x,x_1,\ldots,x_n]}(U^i_{k+1}(b)) \overset{IH}{\equiv}$$

$$\lambda x.u_{[x_n,\ldots,x_{k+i},x_k,\ldots,x_1,x]}(b) = u_{[x_n,\ldots,x_{k+i},x_k,\ldots,x_1]}(a)$$

∎

Lemma 2.40 *Let a, $b \in \Lambda$ and $x_1, \ldots, x_n$, $y_1, \ldots, y_{i-1}$, x distinct variables. Then $u_{[y_1,\ldots,y_{i-1},x_1,\ldots,x_n]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) \equiv (u_{[y_1,\ldots,y_{i-1},x,x_1,\ldots,x_n]}(a))[x := u_{[x_1,\ldots,x_n]}(b)]$.*

Proof. By induction on a. Let us denote $\overline{x} = x_n, \ldots, x_1$ and $\overline{y} = y_{i-1}, \ldots, y_1$.
We study the two interesting cases:

1. $a = \mathtt{j}$ **:** If $j < i$ then $u_{[\overline{x},\overline{y}]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) = y_j = (u_{[\overline{x},x,\overline{y}]}(a))[x := u_{[x_1,\ldots,x_n]}(b)]$.
If $j > i$ then $u_{[\overline{x},\overline{y}]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) = x_{j-i} = (u_{[\overline{x},x,\overline{y}]}(a))[x := u_{[x_1,\ldots,x_n]}(b)]$.
If $j = i$ then

$$u_{[\overline{x},\overline{y}]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) = u_{[\overline{x},\overline{y}]}(U^i_0(b)) \overset{L\,2.39}{\equiv}$$

$$u_{[x_1,\ldots,x_n]}(b) = (u_{[\overline{x},x,\overline{y}]}(a))[x := u_{[x_1,\ldots,x_n]}(b)].$$

2. $a = \lambda d$ **:** We choose $z \neq x$ to obtain:

$$u_{[\overline{x},\overline{y}]}(a\{\!\{\mathtt{i} \leftarrow b\}\!\}) = \lambda z.u_{[\overline{x},\overline{y},z]}(d\{\!\{\mathtt{i+1} \leftarrow b\}\!\}) \overset{IH}{\equiv}$$

$$\lambda z.(u_{[\overline{x},x,\overline{y},z]}(d))[x := u_{[x_1,\ldots,x_n]}(b)] =$$

$$(u_{[\overline{x},x,\overline{y}]}(a))[x := u_{[x_1,\ldots,x_n]}(b)]$$

∎

Lemma 2.41 *Let a, $b \in \Lambda$ such that $FV(a) \subseteq \{1, \ldots, n\}$. If $a \rightarrow_\beta b$ then $u_{[x_1,\ldots,x_n]}(a) \rightarrow_\beta u_{[x_1,\ldots,x_n]}(b)$.*

Proof. Remark that, since $FV(b) \subseteq FV(a)$, as can easily be checked, $u_{[x_1,\ldots,x_n]}(b)$ is well defined.
The proof of the lemma is by induction on a. The interesting case is when a is an application and the reduction takes place at the root.
Therefore, let $a = (\lambda d)e$ and $b = d\{\!\{\mathtt{1} \leftarrow e\}\!\}$. We have

$$u_{[x_1,\ldots,x_n]}(a) = (\lambda x.u_{[x,x_1,\ldots,x_n]}(d))u_{[x_1,\ldots,x_n]}(e) \rightarrow_\beta$$

$$(u_{[x,x_1,\ldots,x_n]}(d))[x := u_{[x_1,\ldots,x_n]}(e)] \overset{L\,2.40}{\equiv} u_{[x_1,\ldots,x_n]}(e\{\!\{\mathtt{1} \leftarrow d\}\!\}) = u_{[x_1,\ldots,x_n]}(b)$$

∎

Theorem 2.42 *Let a, $b \in \Lambda$, if $a \rightarrow_\beta b$ then $u(a) \rightarrow_\lambda u(b)$.*

Proof. It is an immediate consequence of the previous lemma. ∎
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 2.43 *For every $a \in \Lambda$ we have $w(u(a)) = a$.*

Proof. It is enough 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 α-equivalence: $u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) \equiv A$.

Lemma 2.44 *For every $A \in \Lambda_V$ we have $u(w(A)) \equiv A$.*

Proof. It is enough to show that if $FV(A) \subseteq \{x_1, \ldots, x_n\}$ and $x_1, \ldots, x_n$ are distinct variables, then $u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) \equiv A$. We do this by induction on A. The usual two interesting cases are:

$A = x_i$ **:** Since $x_1, \ldots, x_n$ are distinct variables, we have:

$$u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) = u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(x_i)) = u_{[x_1,\ldots,x_n]}(\mathtt{i}) = x_i = A$$

$A = \lambda x.B$ **:** By VC we can assume x distinct from $x_1, \ldots, x_n$. We have:

$$u_{[x_1,\ldots,x_n]}(w_{[x_1,\ldots,x_n]}(A)) = u_{[x_1,\ldots,x_n]}(\lambda w_{[x,x_1,\ldots,x_n]}(B) =$$

$$\lambda x.u_{[x,x_1,\ldots,x_n]}(w_{[x,x_1,\ldots,x_n]}(B)) \overset{IH}{\equiv} \lambda x.B$$

∎

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

Corollary 2.45 *The classical λ-calculus $(\Lambda_V, \rightarrow_\lambda)$ and the λ-calculus à la de Bruijn $(\Lambda, \rightarrow_\beta)$ are isomorphic.*

Theorem 2.46 *The λ-calculus à la de Bruijn is confluent.*

Proof. The confluence of the classical λ-calculus (cf. [2] Thm. 3.2.8) is transportable, via the isomorphism, to the λ-calculus à la de Bruijn.

A proof which does not use the mentioned isomorphism is given in [26] (Corollary 3.6) as a corollary of a more general result concerning the $\lambda\sigma$-calculus. ∎

2.5 The λs-calculus

We recall here the λs-calculus and remind the origin of its rules. We shall follow the same intuition to formulate the rules of the λt-calculus.

The idea is to handle explicitly the meta-operators defined in Definitions 2.6 and 2.7. Therefore, the syntax of the λs-calculus is obtained by adding to the syntax of the λ-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 σ^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 replaced 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 replaced.

Definition 2.47 *The set of* terms of the λs-calculus*, denoted* Λs *, is given as follows:*

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

We take a, b, c to range over Λs. A term containing neither σ's nor φ's is called a pure term.

The λs-calculus should carry out, besides β-reduction, the computations of updating and substitution explicitly. For that reason it contains, besides the rule mimicking the β-rule (σ*-generation*), a set of rules which are the equations in Definitions 2.6 and 2.7 orientated from left to right.

Definition 2.48 *The* λs-calculus *is the reduction system* $(\Lambda s, \to_{\lambda s})$*, where* $\to_{\lambda s}$ *is the least compatible reduction on* Λs *generated by the rules given below:*

σ*-generation*	$(\lambda a)\, b$	$\longrightarrow$	$a\, \sigma^1\, b$
σ*-*λ*-transition*	$(\lambda a)\, \sigma^i b$	$\longrightarrow$	$\lambda(a\, \sigma^{i+1}\, b)$
σ*-app-transition*	$(a_1\, a_2)\, \sigma^i b$	$\longrightarrow$	$(a_1\, \sigma^i b)\,(a_2\, \sigma^i b)$
σ*-destruction*	$\mathtt{n}\, \sigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \textit{if } \ n > i \\ \varphi^i_0\, b & \textit{if } \ n = i \\ \mathtt{n} & \textit{if } \ n < i \end{cases}$
φ*-*λ*-transition*	$\varphi^i_k(\lambda a)$	$\longrightarrow$	$\lambda(\varphi^i_{k+1}\, a)$
φ*-app-transition*	$\varphi^i_k(a_1\, a_2)$	$\longrightarrow$	$(\varphi^i_k\, a_1)\,(\varphi^i_k\, a_2)$
φ*-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 λs to denote this set of rules. The calculus of substitutions associated with the λs-calculus is the rewriting system whose rules are $\lambda s - \{\sigma\text{-generation}\}$ and we call it s-calculus.

The main results concerning the λs-calculus are (see [19] for proofs):

Theorem 2.49 *The λs-calculus is confluent on Λs.*

Theorem 2.50 (PSN of λs) *Every λ-term which is strongly normalising in the classical λ-calculus is also strongly normalising in the λs-calculus.*

2.6 The $\lambda\sigma$- and $\lambda\upsilon$-calculi

We recall now the terms and rules of $\lambda\sigma$ and $\lambda\upsilon$, so that the reader not familiar with them could refer to this subsection before reading Section 6. For the motivations and properties of $\lambda\sigma$ see [1, 7, 26] and for $\lambda\upsilon$ refer to [3].

Definition 2.51 *The syntax of the $\lambda\sigma$-*calculus *is given by:*

$$\begin{array}{ll} \textbf{Terms} & \Lambda\sigma^t ::= \mathtt{1} \mid \Lambda\sigma^t\Lambda\sigma^t \mid \lambda\Lambda\sigma^t \mid \Lambda\sigma^t[\Lambda\sigma^s] \\ \textbf{Substitutions} & \Lambda\sigma^s ::= id \mid \uparrow \mid \Lambda\sigma^t \cdot \Lambda\sigma^s \mid \Lambda\sigma^s \circ \Lambda\sigma^s \end{array}$$

*The set, denoted $\lambda\sigma$, of rules of the $\lambda\sigma$-*calculus *is the following:*

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b \cdot id]$
(VarId)	$\mathtt{1}\,[id]$	$\longrightarrow$	$\mathtt{1}$
(VarCons)	$\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	a
(App)	$(a\,b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\mathtt{1} \cdot (s \circ \uparrow)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(IdL)	$id \circ s$	$\longrightarrow$	s
(ShiftId)	$\uparrow \circ\, id$	$\longrightarrow$	$\uparrow$
(ShiftCons)	$\uparrow \circ\, (a \cdot s)$	$\longrightarrow$	s
(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(Ass)	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow$	$s_1 \circ (s_2 \circ s_3)$

*The set of rules of the σ-*calculus *is $\lambda\sigma - \{(Beta)\}$. We let $a, b, c, \ldots$ range over $\Lambda\sigma^t$ and $s, t, \ldots$ range over $\Lambda\sigma^s$. For $s \in \Lambda\sigma^s$, we define $s^0 = id, s^1 = s$ and $s^{n+1} = s \circ s^n$.*

Definition 2.52 *The syntax of the $\lambda\upsilon$-*calculus *is given by:*

$$\begin{array}{ll} \textbf{Terms} & \Lambda\upsilon^t ::= \mathbb{N} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s] \\ \textbf{Substitutions} & \Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t/ \end{array}$$

*The set, denoted $\lambda\upsilon$, of rules of the $\lambda\upsilon$-*calculus *is the following:*

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b/]$
(App)	$(a\,b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(FVar)	$\mathtt{1}\,[a/]$	$\longrightarrow$	a
(RVar)	$\mathtt{n}+\mathtt{1}\,[a/]$	$\longrightarrow$	$\mathtt{n}$
(FVarLift)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(RVarLift)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}[s][\uparrow]$
(VarShift)	$\mathtt{n}[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$

We let $a, b, c, \ldots$ range over $\Lambda\upsilon^t$ and $s, t, \ldots$ range over $\Lambda\upsilon^s$. For $a \in \Lambda\upsilon^t$ and $s \in \Lambda\upsilon^s$, we define $\Uparrow^0(s) = s$, $a[s]^0 = a$, $\Uparrow^i(s) = \Uparrow(\Uparrow \ldots (\Uparrow(s) \ldots))$ and $a[s]^i = a[s][s] \ldots [s]$ (i times).

3 Another presentation of the λ-calculus à la de Bruijn

In Definition 2.7 we have defined $\mathtt{i}\{\!\{\mathtt{i} \leftarrow b\}\!\} = U_0^i(b)$, but there is another choice: instead of updating b when performing the substitution we can make partial updatings of b, each time the substitution operator traverses a λ 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 3.1 *The* new updating functions $V_k : \Lambda \rightarrow \Lambda$ *for $k \geq 0$ are defined inductively as follows (compare with Definition 2.6):*

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

Definition 3.2 *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 2.7):*

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

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

Notation 3.3 *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, $V_k^0(a) = a$.*

Note that, in Definition 2.7 we have defined $\mathtt{i}\{\!\{\mathtt{i} \leftarrow b\}\!\} = U_0^i(b)$, but now, with Definition 3.2, $\mathtt{i}[\![\mathtt{i} \leftarrow b]\!] = b$ because we have already performed (in partial steps) the substitution. In order to connect those two notions of substitutions, we needed the above notation (another reason for the above notation was necessitated by the proofs of the various lemmas below):[4]

Lemma 3.4 *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. ∎

[4]Note that when one reaches a leaf i, $\{\!\{\leftarrow\}\!\}$ necessitates an immediate updating by i whereas $[\![\leftarrow]\!]$ requires no updating because updating occurred at each level of descent into the tree. This explains Lemma 3.4.

Remark 3.5 *As a particular case of Lemma 3.4.2 we have* $a\{\!\{\mathtt{1} \leftarrow b\}\!\} = a[\![\mathtt{1} \leftarrow b]\!]$ *and hence we can describe* β*-reduction using the new meta-substitution functions as:*

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

Unfortunately Lemma 3.4 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 3.6 *For* $k \geq 0$ *we have* $V_k(V_0^k(c)) = V_0^{k+1}(c)$.

Proof. By Lemma 2.9, $U_k^2(U_0^{k+1}(c)) = U_0^{k+2}(c)$. Now, use Lemma 3.4.1. ∎

The following lemma, though related to Lemma 2.8, cannot be deduced directly from it, as we did for the previous lemma.

Lemma 3.7 *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 induction on the structure of a. ∎

Again, the next lemma, though related to Lemma 2.10, cannot be deduced from it.

Lemma 3.8 *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 3.6 for the case $a = \mathtt{n}$. ∎

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

Lemma 3.9 (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 3.8 is necessary for the case $a = \lambda d$ and Lemma 3.7 settles the case $a = \mathtt{n}+\mathtt{1}$. ∎

Finally, the following lemma ensures the good passage of the β-rule through the new meta-substitutions and updatings. It is crucial for the proof of the confluence of λt.

Lemma 3.10 *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 3.4.1 and Lemma 2.15.1

2. Induction on a using 1 above.

3. Induction on a. The interesting case is $a = (\lambda d)e$ and $b = d[\![\mathtt{1} \leftarrow e]\!]$:

$$((\lambda d)e)[\![\mathtt{i} \leftarrow c]\!] = (\lambda(d[\![\mathtt{i}+\mathtt{1} \leftarrow V_0(c)]\!]))(e[\![\mathtt{i} \leftarrow c]\!]) \to_\beta$$

$$(d[\![\mathtt{i}+\mathtt{1} \leftarrow V_0(c)]\!])[\![\mathtt{1} \leftarrow e[\![\mathtt{i} \leftarrow c]\!]]\!] \overset{L\,3.9}{=} (d[\![\mathtt{1} \leftarrow e]\!])[\![\mathtt{i} \leftarrow c]\!]$$

∎

4 The λt-calculus

Now, we shall handle explicitly the new meta-operators defined in Definitions 3.1 and 3.2. Therefore, the syntax of the λt-calculus is obtained by adding to the syntax of the λ-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 ς^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 replaced by the *already updated* term b.
- $\{\theta_k\}_{k\geq 0}$: this family is meant to denote the new updating functions.

Definition 4.1 *The set of* terms of the λt-calculus*, denoted* Λt*, is given as follows:*

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

We take a, b, c to range over Λt. A term containing neither ς's nor θ's is called a pure term*. By $\theta_k^i a$ for $i \geq 1$, we mean $\theta_k(\theta_k(\ldots(\theta_k a)))$ (i θ_k-operators) and $\theta_k^0 a$ means a.*

The λt-calculus should carry out, as the λs-calculus, besides β-reduction, the computations of updating and substitution explicitly. For that reason we include, besides the rule mimicking the β-rule (ς-*generation*), a set of rules which are the equations in the Definitions 3.1 and 3.2 orientated from left to right.

Definition 4.2 *The* λt-calculus *is the reduction system* $(\Lambda t, \rightarrow_{\lambda t})$*, where* $\rightarrow_{\lambda t}$ *is the least compatible reduction on* Λt *generated by the rules given below:*

ς-*generation*	$(\lambda a)\, b$	$\longrightarrow$	$a\,\varsigma^1\, b$
ς-λ-*transition*	$(\lambda a)\,\varsigma^i b$	$\longrightarrow$	$\lambda(a\,\varsigma^{i+1}\,\theta_0(b))$
ς-*app-transition*	$(a_1\, a_2)\,\varsigma^i b$	$\longrightarrow$	$(a_1\,\varsigma^i b)\,(a_2\,\varsigma^i b)$
ς-*destruction*	$\mathtt{n}\,\varsigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \textit{if} \;\; n > i \\ b & \textit{if} \;\; n = i \\ \mathtt{n} & \textit{if} \;\; n < i \end{cases}$
θ-λ-*transition*	$\theta_k(\lambda a)$	$\longrightarrow$	$\lambda(\theta_{k+1}\, a)$
θ-*app-transition*	$\theta_k(a_1\, a_2)$	$\longrightarrow$	$(\theta_k\, a_1)\,(\theta_k\, a_2)$
θ-*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 λt to denote this set of rules. The calculus of substitutions associated with the λt-calculus is the rewriting system whose rules are $\lambda t - \{\varsigma$-generation$\}$ and we call it t-calculus. *We will use obvious abbreviations of these rules when $\rightarrow$ is used. E.g. we write $\rightarrow_{\varsigma-gen}$.*

The main difference between λt and λs can be summarized as follows: the λt-calculus generates a partial updating when a substitution is evaluated on an abstraction (i.e. introduces an operator θ_0 in the ς-λ-*transition* rule) whereas the λs-calculus produces a global updating when performing substitutions (i.e. introduces a φ_0^i operator in the *σ-destruction* rule, case $n = i$).

The λt-calculus shares this mechanism of partial updatings with the $\lambda\sigma$-calculi and their descendants $\lambda\upsilon$ and $\lambda\zeta$ since all of them introduce an updating operator in their *substitution-abstraction-transition* rule (see rule (Abs) in both $\lambda\sigma$ and $\lambda\upsilon$ above).

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

Theorem 4.3 (SN and confluence of t) *The t-calculus is SN and confluent on Λt. 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 ς-λ-*transition* then $W_1(a) > W_1(b)$.

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

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

Since there are no critical pairs, the theorem of Knuth-Bendix (cf. [17, 15]) applies trivially to yield the local confluence of the t-calculus.

Finally, Newman's lemma, which states that every strongly normalising and locally confluent relation is confluent (cf. [2], Proposition 3.1.25), provides the confluence of the t-calculus. ∎

Lemma 4.4 *The set of t-normal forms is exactly Λ.*

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 Λ is a t-nf. ∎

Lemma 4.5 *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 λt-calculus: the good passage of the ς-generation rule to the t-normal forms.

Lemma 4.6 *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 $b = e\,\varsigma^1 d$:

$$t((\lambda e)d) = (\lambda t(e))(t(d)) \to_\beta t(e)[\![\mathtt{1} \leftarrow t(d)]\!] \overset{L\,4.5}{=} t(e\,\varsigma^1 d)$$

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

$$t(c\,\varsigma^i d) \overset{L\,4.5}{=} t(c)[\![\mathtt{i} \leftarrow t(d)]\!] \overset{IH\ \&\ L\,3.10.3}{\twoheadrightarrow_\beta} t(e)[\![\mathtt{i} \leftarrow t(d)]\!] \overset{L\,4.5}{=} t(e\,\varsigma^i d)$$

If the reduction takes place within d, lemma 3.10.2 applies.

$a = \theta_k c$ **:** The reduction must take place within c. Use Lemma 4.5 and Lemma 3.10.1.

∎

Now, the following corollaries are immediate.

Corollary 4.7 *Let* $a, b \in \Lambda t$ *, if* $a \twoheadrightarrow_{\lambda t} b$ *then* $t(a) \twoheadrightarrow_\beta t(b)$ *.*

Corollary 4.8 (Soundness) *Let* $a, b \in \Lambda$ *, if* $a \twoheadrightarrow_{\lambda t} b$ *then* $a \twoheadrightarrow_\beta b$ *.*

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

Finally, before proving confluence, we verify that the λt-calculus is powerful enough to simulate β-reduction.

Lemma 4.9 (Simulation of β-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 $b = c[\![\mathtt{1} \leftarrow d]\!]$:

$$(\lambda c)d \to_{\varsigma-gen} c\varsigma^1 d \twoheadrightarrow_t t(c\varsigma^1 d) \overset{L4.5}{=} t(c)[\![\mathtt{1} \leftarrow t(d)]\!] \overset{c,d \in \Lambda}{=} c[\![\mathtt{1} \leftarrow d]\!]$$ ∎

Theorem 4.10 (Confluence of λt) *The λt-calculus is confluent on Λt.*

Proof. We interpret the λt-calculus into the λ-calculus via t-normalisation. We have:

$$\begin{tikzcd}
 & b \arrow[rr, two heads, "t"] & & t(b) \arrow[dr, two heads, "\beta"] & \\
a \arrow[ur, two heads, "\lambda t"] \arrow[rr, two heads, "t"] \arrow[dr, two heads, "\lambda t"'] & & t(a) \arrow[ur, two heads, "\beta"] \arrow[dr, two heads, "\beta"'] & \text{Thm. 2.46} & d \\
 & c \arrow[rr, two heads, "t"] & & t(c) \arrow[ur, two heads, "\beta"'] &
\end{tikzcd}$$

The existence of the arrows $t(a) \twoheadrightarrow_\beta t(b)$ and $t(a) \twoheadrightarrow_\beta t(c)$ is guaranteed by Corollary 4.7. We can close the diagram thanks to the confluence of the λ-calculus and finally Lemma 4.9 ensures $t(b) \twoheadrightarrow_{\lambda t} d$ and $t(c) \twoheadrightarrow_{\lambda t} d$ proving thus CR for the λt-calculus. ∎

5 Interpretation of λt into $\lambda\mathtt{x}$

The function that interprets λt into $\lambda\mathtt{x}$ is an extension of the function $u : \Lambda \to \Lambda_V$ (cf. Definition 2.38). Here we stop to explain the work of Rose in his thesis [28]. In [28], Rose provides a translation from $\lambda\upsilon$ to $\lambda\mathtt{x}$ which he shows to satisfy similar propertiels s to those of Theorem 5.9 below. Rose uses that translation to deduce PSN of $\lambda\upsilon$ from PSN of $\lambda\mathtt{x}$. He then abstracts a sufficient condition on such a translation, called explicit naming, which guarantees PSN. He uses this abstraction result to give the proofs of PSN of λs and $\lambda\chi$ of [23] directly from the proof of PSN for $\lambda\mathtt{x}$. We follow a similar line here where we give a translation of λt into $\lambda\mathtt{x}$ (which preserves reduction) and we use this translation to show PSN of λt from PSN of $\lambda\mathtt{x}$. The difference between our approach and that of [28] lies in the manipulation of the variable lists used to define the interpretation into $\lambda\mathtt{x}$. In the case of [28], those variable lists are used to interpret λs which has different substitution and updating mechanisms than λt. Hence, it is inevitable that the manipulation of the variable lists be different in both approaches. It should also be noted that there is another approach ([16]) that interprets substitution calculi with de Bruijn indices in the classical λ-calculus (without substitutions) and which inevitably manipulates variable lists but in a manner different to that of this paper and of [28].

In order to proceed, we must extend the notion of free variable.

Again, in the following (Definitions 5.1, 5.2 and 5.6), in order to keep the notation lighter, we use the same notation for the original notions and the extended ones. The context will be always clear enough to avoid confusions.

Definition 5.1 *The* set of free variables *of a term in Λt is defined by extending Definition 2.18 as follows:*

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

Definition 5.2 *Let $d \in \Lambda t$ such that $FV(d) \subseteq \{1, \ldots, n\}$ and let $x_1, \ldots, x_n \in V$. We define $u_{[x_1,\ldots,x_n]}(d)$ by extending Definition 2.37 as follows:*

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

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

Note that for the case $a\,\varsigma^i b$ where $n \ge i$, x is inserted between x_{i-1} and x_i (not x_{i+1}).

In order to check that Definition 5.2 is correct, the following remark, whose proof is easy, is needed.

Remark 5.3 *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 $n \ge i$ 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 5.4 *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 5.5 *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 5.6 *We define $u : \Lambda t \to \Lambda\mathbf{x}$ 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 5.7 *If $a \in \Lambda t$, $FV(a) \subseteq \{1, \ldots, n\}$ and $m > n$ then $u_{[v_1,\ldots,v_n]}(a) \equiv u_{[v_1,\ldots,v_m]}(a)$.*

Proof. Easy induction on a. ∎

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, $u(\theta_1 \mathtt{1}) = v_1 = u(\mathtt{1})$.

Lemma 5.8 *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 2.17. ∎

Theorem 5.9 *Let a, $b \in \Lambda t$.*

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

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

Remark first that Lemma 5.8 guarantees the correct definition of $u_{[x_1,\dots,x_n]}(b)$.

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 ς-λ-*transition*:

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

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

It is this case that shows why the rule σ-λ-*transition* of the λs-calculus had to be changed into the rule ς-λ-*transition* of the λt-calculus if reduction was to be preserved.

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

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

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

If $n > 0$ we have:

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

If $n = 0$ we have:

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

where equality (1) holds because of Lemma 5.7 (with $n = 0$ and $m = 1$) and the fact that $FV(a) = \phi$ (since $FV(a\varsigma^1 b) = \phi$, Remark 5.3 yields $FV(a) = \phi$). ∎

5.1 λt preserves strong normalisation

Using Theorem 5.9 and the PSN of $\lambda\mathbf{x}$, we can show the PSN of λt. 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 5.10 (PSN of λt) *Every λ-term which is strongly normalising in the λ-calculus à la de Bruijn is also strongly normalising in the λt-calculus.*

PROOF. Since $a \in \lambda$-SN, Theorem 2.36 and Lemma 2.43 guarantee that $u(a)$ is strongly normalising in the classical sense. The Preservation Theorem for $\lambda\mathtt{x}$ (see Theorem 2.27) ensures $u(a) \in \lambda\mathtt{x}$-SN.

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

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

be an infinite derivation. Since the t-calculus is SN (see Theorem 4.3), this derivation must contain an infinity of ς-generations:

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

Now, by Theorem 5.9.2 and 5.9.3, we have:

$$u(a) \twoheadrightarrow_{\mathtt{x}} u(a'_1) \to_{\lambda\mathtt{x}} u(a'_2) \twoheadrightarrow_{\mathtt{x}} \ldots \twoheadrightarrow_{\mathtt{x}} u(a'_{2n+1}) \to_{\lambda\mathtt{x}} u(a'_{2n+2}) \twoheadrightarrow_{\mathtt{x}} \ldots$$

and this contradicts the fact that $u(a) \in \lambda\mathtt{x}$-SN. Therefore, $a \in \lambda t$-SN. ∎

6 Comparison with $\lambda\sigma$ and $\lambda\upsilon$

The λt calculus can be interpreted into the $\lambda\sigma$ calculus using a similar translation as the one presented in [19] to interpret the λs-calculus into $\lambda\sigma$. However, in the case of the λt-calculus the interpretation works better: now λt-derivations are preserved (only s-derivations and not λs derivations were preserved by the translation in [19].)

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

Definition 6.1 *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 6.2 *Let $b \in \Lambda\sigma^t$, we define a family of substitutions $(b_k)_{k \geq 1}$ as follows:*
$b_1 = b \cdot id \qquad b_2 = \mathtt{1} \cdot b \cdot \uparrow \qquad \ldots \qquad b_{i+1} = \mathtt{1} \cdot \mathtt{2} \cdot \ldots \cdot \mathtt{i} \cdot b \cdot \uparrow^i \qquad \ldots$

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

Remark 6.3 $\mathtt{1} \cdot (b_i \circ \uparrow) \twoheadrightarrow_\sigma (b[\uparrow])_{i+1}$ *and* $\mathtt{1} \cdot (s_k \circ \uparrow) \twoheadrightarrow_\sigma s_{k+1}$.

Definition 6.4 *The* translation function $T : \Lambda s \to \Lambda\sigma^t$ *is defined by:*
$T(\mathtt{n}) = \mathtt{n} \quad T(a\, b) = T(a)T(b) \quad T(a\, \varsigma^i b) = T(a)[T(b)_i] \quad T(\lambda a) = \lambda(T(a))$
$T(\theta_k a) = T(a)[s_k]$

Theorem 6.5 *If $a \to_{\lambda t} b$ then $T(a) \overset{+}{\twoheadrightarrow}_{\lambda\sigma} T(b)$.*

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:

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

∎

Even if λt is interpreted in $\lambda\sigma$ more faithfully than λs (the *σ-generation* rule translates (cf. [19]) into a $\lambda\sigma$-equivalence rather than a derivation), no reasonable translation of λt 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 [19] $\mathtt{1}_2$ was defined as $\mathtt{1} \cdot \mathtt{1}[\uparrow] \cdot \uparrow$, and this $\lambda\sigma$-substitution is available 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 β-reduction in λt and $\lambda\upsilon$. We choose now $\lambda\upsilon$ instead of $\lambda\sigma$ because $\lambda\sigma$ is incomparable to any of the calculi $\lambda\upsilon$, λt and λs as we have shown in [21]. On the other hand, here we show that λt is more efficient than $\lambda\upsilon$. In other words, we are going to prove that β-simulation in λt (one step *ς-generation* followed by t-derivation to normal form) is more efficient than β-simulation in $\lambda\upsilon$ (one step *Beta*, written $\rightarrow_B$ followed by υ-derivation to normal form).

We begin by introducing a set of terms Λ_θ 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 Λ_θ, but the introduction of Λ_θ is necessary since it provides a strong induction hypothesis to prove the auxiliary results needed.

Definition 6.6 $\Lambda_\theta ::= \mathbb{N} \mid \Lambda_\theta\Lambda_\theta \mid \lambda\Lambda_\theta \mid \theta_k\Lambda_\theta$ *, where* $k \geq 0$*. The* length *of terms in* Λ_θ *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 6.7 *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 $t(\theta_m b)$) 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 6.8 *If* $a \in \Lambda_\theta$ *and* $a \rightarrow_t b$ *then* $b \in \Lambda_\theta$*.*

Proof. Easy induction on a. ∎

Definition 6.9 *We define* $M : \Lambda_\theta \rightarrow \mathbb{N}$ *by induction as follows:*

$$\begin{array}{ll} M(\mathtt{n}) = 1 & M(ab) = M(a) + M(b) + 1 \\ M(\lambda a) = M(a) + 1 & M(\theta_k a) = M(t(\theta_k a)) + M(a) \end{array}$$

Remark that the definition is correct thanks to Remark 6.7.

Lemma 6.10 *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 $P(a) = (W_1(a), W_2(a))$ used to prove SN for the t-calculus (see proof of Theorem 4.3). 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 $b \rightarrow b'$, 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 6.11 *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 $a \in \Lambda_\theta$, using Remark 6.7, that $M(t(a)) = M(t(\theta_k a))$, then use this result to prove, by induction on $j \geq 1$ that $M(t(a)) = M(t(\theta^j_k a))$. Use now Definition 6.9 and the two previous results to show, by induction on $l \geq 1$, that $M(\theta^l_k(a)) = lM(t(a)) + M(a)$. Finally, use Lemma 6.10 and the last result with $l = i - 1$ to prove the corollary. Remark that it is in this proof that the hypothesis $a \in \Lambda_\theta$ is essential and hence the necessity of Definition 6.6. ∎

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

Definition 6.12 $\Lambda_\uparrow ::= \ \mathbb{N} \mid \Lambda_\uparrow\Lambda_\uparrow \mid \lambda\Lambda_\uparrow \mid \Lambda_\uparrow[\Uparrow^k(\uparrow)]$ *, 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 6.13 *Let $a \in \Lambda_\uparrow$ and $k \geq 0$, then $L_\uparrow(a) \geq L_\uparrow(\upsilon(a[\Uparrow^k(\uparrow)]))$.*

Remark 6.14 *If $a \in \Lambda_\uparrow$ and $a \to_\upsilon b$ then $b \in \Lambda_\uparrow$.*

Definition 6.15 *For $k \geq 0$, we define $M_k : \Lambda_\theta \to \mathbb{N}$ as follows:*

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

Lemma 6.16 *For $a \in \Lambda_\uparrow$, all the υ-derivations of $a[\Uparrow^k(\uparrow)]$ to its υ-nf have length $M_k(a)$.*

Proof. By induction on the weight used to show SN for the υ-calculus (cf. [3]) and case analysis. ∎

Corollary 6.17 *For $a \in \Lambda_\uparrow$, all the υ-derivations of $a[\Uparrow^k(\uparrow)]^i$ to its υ-normal form have the same length, namely $(i-1)M_k(\upsilon(a)) + M_k(a)$.*

Definition 6.18 *Let $a, b \in \Lambda$ and $i \geq 0$, we define $P_i(a, b)$ by induction on a:*

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

Lemma 6.19 *Let $a, b \in \Lambda$ and $i \geq 0$, all the υ-derivations of $a[\Uparrow^i(b/)]$ to its υ-nf have the same length, namely $P_i(a, b)$.*

Proof. Easy induction on $a \in \Lambda$. Remark that for $a = \mathtt{n}$ there is only one derivation whose length is easy to compute. When $n = i + 1$, use Corollary 6.17. ∎

Lemma 6.20 *Let $a, b \in \Lambda$ and $i \geq 0$, there exists a derivation of $a \varsigma^{i+1} (\theta_0^i b)$ 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 $M_0(b) \geq M(b)$ (induction on $b \in \Lambda$) and Corollary 6.11. ∎

Theorem 6.21 *β-simulation is more efficient in λt than in $\lambda \upsilon$.*

Proof. We prove that for every $a \in \Lambda$ and every $\lambda\upsilon$-derivation $a \to_B b \twoheadrightarrow_\upsilon^m \upsilon(b)$ there exists $n \leq m$ such that $a \to_{\varsigma-gen} c \twoheadrightarrow_t^n t(c)$ by induction on a.

The interesting case is $a = (\lambda d)e \to_B d[e/] \twoheadrightarrow^m \upsilon(d[e/])$. By Lemma 6.19 we know that $m = P_0(d, e)$ and Lemma 6.20 gives a derivation $d \varsigma^1 e \twoheadrightarrow_t^n t(d \varsigma^1 e)$ such that $n \leq P_0(d, e)$.

Remark that there are an infinity of cases for which the inequality is strict. For instance, let us consider the term $(\lambda\lambda \ldots \lambda \mathtt{n})a$ with m λ'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 β-reduction in $\lambda\upsilon$, whereas only $m + 1$ reductions are sufficient in λt. Remark that for $m > n$ the number of reductions needed in $\lambda\upsilon$ is also strictly greater than the number needed in λt. ∎

7 About extensions on open terms

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

Let us recall that such an extension was successful for λs and gave rise to the confluent calculus λs_e (cf. [22]).

Definition 7.1 *The set of* open terms*, denoted Λt_{op}, is given as follows:*

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

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

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

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

and $(X \varsigma^1 Y)\varsigma^1 \mathtt{1}$ 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 λs, we found that, since the term $((\lambda X)\sigma^1 \mathtt{1})(Y \sigma^1 \mathtt{1}) \twoheadrightarrow (X \sigma^2 \mathtt{1})\sigma^1 (Y \sigma^1 \mathtt{1})$, the solution to the problem was at hand if one had in mind the properties of meta-substitutions and updating functions of the λ-calculus in the Bruijn notation (cf. Lemmas 2.8 - 2.13). These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to λs give origin to a rewriting system which happens to be locally confluent (cf. [20]). For instance, the rule corresponding to the Meta-substitution lemma (Lemma 2.11) is the *σ-σ-transition* rule given below.

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

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

Following the same method we can try an orientation of the equality given in Lemma 3.9 to find our ς-ς-*transition* rule:

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

Remark that in the σ-σ-*transition* rule no updating operator appears. The presence of updating operators in the ς-ς-transition rule gives rise to undesirable critical pairs. For instance:

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

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

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

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

Therefore, the λt-calculus does not seem to possess a reasonable extension on open terms. We do not consider this as a negative property of λt. After all, $\lambda\upsilon$ does not have a confluent extension on open terms. Moreover, there has never been given a confluent extension of substitution calculi using variable names, although it seems that some progress is being made in this area through a calculus proposed by Laing and Rose that has been shown to be locally confluent and whose confluence is expected by its inventors.

8 Conclusion

Even if the λt-calculus cannot be extended to a confluent extension on open terms (of the calculi mentioned in the Introduction, only the λs-calculus, the $\lambda\sigma_\Uparrow$-calculus 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 three reasons:

1. It can be related to $\lambda\mathtt{x}$, as we have shown in this paper, via a translation which is an extension of the classical isomorphism between the classical λ-calculus and its de Bruijn version.
2. While being a calculus à la λs, 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 λs- and the $\lambda\sigma$-style.
3. It simulates β-reduction more efficiently than $\lambda\upsilon$.

One of the questions we raised in the Introduction is still unanswered, namely if the $\lambda\mathtt{x}$-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 interpretation in the other direction have failed and we conclude this paper by pointing out the problems that arise when trying to define such an interpretation, i.e. a translation of $\lambda\mathsf{x}$ into λt.

Now the question is how to extend the functions $w_{[x_1,\ldots,x_n]}$ given in Definition 2.30. 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 λ'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 could 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):

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

The reader can easily check that with this definition reduction is now preserved for the *σ-λ-transition* rule of the $\lambda\mathsf{x}$-calculus (assuming that a lemma analogous to Lemma 2.34 should hold for the operators θ_k). But unfortunately the *σ-generation* rule is the one that fails now.

Therefore the question of the existence of an extension of w preserving reduction remains open. Furthermore, it is not clear what calculus of explicit substitutions à la de Bruijn could be isomorphic to $\lambda\mathsf{x}$. 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.

Furthermore, we mention that work on the connections between variable names and de Bruijn indices in the context of explicit substitution calculi is a topic that is attracting new interest amongst the community of researchers working on explicit substitutions. In particular, we mention the work of Pierre Lescanne on explicit substitutions and explicit alpha conversion [23]. This work is complementary to ours and we believe that both our work and that of Lescanne may lead to theories of explicit substitutions that have the advantages desired from the point of view of controlling substitutions and of being simple for human users to work with.

Finally, it might be questioned why we proposed another calculus of explicit substitutions (λt) if it does not satisfy more properties than previous calculi. For example, λt does not seem to possess a confluent extension on open terms whereas λs and $\lambda\sigma$ do. Here we reply that as the area of explicit substitutions still has many open problems, research that enables relating different lines in the area is necessary to increase our understanding in the subject and this may lead to more collaborations between researchers that belong to different schools of substitutions. For example, λs and $\lambda\upsilon$ were the first calculi of explicit substitutions that satisfy PSN. $\lambda\mathtt{x}$ was introduced by Rose in [27] and was shown in [6] to satisfy the same properties of $\lambda\upsilon$ and λs (namely simulation of β-reduction, confluence on closed terms and PSN). Work on λs was taken further and it was shown that λs has a confluent extension on open terms. Up to now, this is not the case for either $\lambda\mathtt{x}$ or $\lambda\upsilon$ (although it is the case for $\lambda\sigma$, but $\lambda\sigma$ does not satisfy PSN). Recently, a calculus was proposed in [12] that is confluent on closed terms, satisfies PSN and also allows composition of substitutions (λs, $\lambda\mathtt{x}$, $\lambda\upsilon$ do not allow composition of substitutions, λs_e and $\lambda\sigma$ do but they do not enjoy PSN as was shown by Melliès in [24] for $\lambda\sigma$ and Guillome in [13] for λs_e). The calculus of [12] however has not been extended to a confluent calculus on open terms. With all this progress and all the remaining open problems, it is vital to bridge the different approaches of explicit substitutions. λt attempts to build such a bridge.

1. λt is a calculus à la λs yet it works with partial updating à la $\lambda\sigma$ and $\lambda\upsilon$. In fact, we showed in Theorem 6.5 that λt can be interpreted in $\lambda\sigma$ more faithfully than λs can be interpreted in $\lambda\sigma$ in the sense that the σ-generation of λs translates into a $\lambda\sigma$-equivalence rather than a derivation.
2. Another point worth investigating in the future is the efficiency of the simulation of β-reduction. In [21] we show that $\lambda\upsilon$ and $\lambda\sigma$ are incomparable in the sense that some β-simulations are shorter in $\lambda\upsilon$ and others are shorter in $\lambda\sigma$. λt on the other hand, has been shown in this article to simulate β-reduction more efficiently than $\lambda\upsilon$. We believe that similar arguments can be used to investigate the efficiency of different calculi.
3. The problem of finding an isomorphism between calculi of explicit substitutions with names and with de Bruijn indices remains, but λt has clarified where the problem occurs in finding the isomorphism as we have explained above. It remains a puzzle whether an isomorphism (as defined in our paper) exists between the named versus the de Bruijn versions of explicit substitution calculi. The finding of such an isomorphism will be a very useful step because calculi with variable names are more user-friendly than those with de Bruijn indices.
4. As explained before, most calculi of explicit substitutions (old and new) perform updating partially. Furthermore, the λs-style of substitution differs from the $\lambda\sigma$-style, not only in the partial versus total updating, but also in the fact that λs does not allow two sorts of expressions (terms and substitutions). For this reason, it is useful to find a calculus of explicit substitutions with one sort of expressions (terms) and which updates partially like $\lambda\sigma$. λt is such a calculus.
5. λt is also interesting because it highlights the problem of extending a calculus à la λs but with partial updating into a confluent extension on open terms. We have not shown in this paper the impossibility of such an extension, but Bruno Guillome is studying it and it seems unlikely that such an extension exists (follow-

ing Guillome). Here, we point out that a negative result is not necessarily a bad result. A negative result often clarifies the picture and rules out some possibilities.

6. Both calculi à la $\lambda\sigma$ (e.g. $\lambda\sigma$, $\lambda\upsilon$, $\lambda\sigma_{\Uparrow}$, $\lambda\zeta$) and à la λs (e.g. λs, λs_e and λt) have failed so far in being a calculus that satisfies a certain collection of properties. Research on finding such a calculus, has benefitted greatly from previous failures. λt (and not λs) has been the basis of a new calculus λw that is being studied by Kamareddine and Wells as a candidate to being a calculus with the collection of desirable properties. The new calculus λw is much more complex than either λs and λt and only a further study and comparison of the various calculi can clarify why this complexity had to be assumed.

Based on these reasons, we believe that λt is interesting and deserves attention because it is one-step further in clarifying the status at which we stand and in pointing out further directions.

Acknowledgements

This work was carried out under EPSRC grants GR/K25014 and GR/L36963.

References

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

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

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

[4] R. Bloo and H. Geuvers. Explicit Substitution: on the Edge of Strong Normalisation . *Theoretical Computer Science*, 1998. To appear.

[5] R. Bloo. *Preservation of Termination for Explicit Substitutions.* PhD thesis, Technical University of Eindhoven, 1997.

[6] R. Bloo and K. H. Rose. Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection in Proceedings of CSN'95. *Computing Science in the Nederlands*, 1995.

[7] 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.

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

[9] N. G. 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.

[10] 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.

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

[12] Ferreira, Kesner, and Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalisation. *ALP*, 1996.

[13] B. Guillome. The λs_e does not preserve strong normalisation. *Private Communications*, November 1997.

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

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

[16] F. Kamareddine. The Soundness of Explicit Substitutions with Nameless Variables. *International Journal of Foundations of Computer Science*, to appear.

[17] 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.

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

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

[20] F. Kamareddine and A. Ríos. The λs-calculus: its typed and its extended versions. Technical report, Department of Computing Science, University of Glasgow, 1995.

[21] F. Kamareddine and A. Ríos. Efficiency of lambda-calculi with Explicit Substitutions. Technical Report TR-1996-3, University of Glasgow, 1996.

[22] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Functional Programming*, 7(4):395–420, 1997.

[23] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de bruijn's levels. *Proceedings 6th Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science*, 914:294–308, 1995.

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

[25] C. A. Muñoz Hurtado. Confluence and preservation of strong normalisation in an explicit substitutions calculus. *Proceeddings of LICS'96*, pages 440–447, 1996.

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

[27] K.H. Rose. Explicit cyclic substitutions. Semantics Note D-166, DIKU, Universitetsparken 1, DK-2100 København Ø, Denmark, March 1993.

[28] K.H. Rose. *Operational Reduction Models for Functional Programming Languages.* PhD thesis, DIKU (University of Copenhagen), 1996. Rapport Nr 96/1, ISSN 0107-8283.

Received 8 September 1997. Revised 30 April 1998