Relating the $\lambda\sigma$- and λs-Styles of Explicit Substitutions

Fairouz Kamareddine* and Alejandro Ríos†

Abstract

The aim of this article is to compare two styles of Explicit Substitutions: the $\lambda\sigma$- and λs-styles. We start by introducing a criterion of adequacy to simulate β-reduction in calculi of explicit substitutions and we apply it to several calculi: $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, λv, λs, λt and λu. The latter is presented here for the first time and may be considered as an adequate variant of λs. By doing so, we establish that calculi à la λs are usually more adequate at simulating β-reduction than calculi in the $\lambda\sigma$-style. In fact, we prove that λt is more adequate than λv and that λu is more adequate than λv, $\lambda\sigma_{\Uparrow}$ and λs. We also give counterexamples to show that all other comparisons are impossible according to our criterion.

Our next step consists in presenting the $\lambda\omega$ and $\lambda\omega_e$ calculi, the two-sorted (`term` and `substitution`) versions of the λs (cf. [KR95]) and λs_e (cf. [KR97]) calculi, respectively. We establish an isomorphism between the λs-calculus and the `term` restriction of the $\lambda\omega$-calculus, which extends to an isomorphism between λs_e and the `term` restriction of $\lambda\omega_e$. Since the $\lambda\omega$ and $\lambda\omega_e$ calculi are given in the style of the $\lambda\sigma$-calculus (cf. [ACCL91]) they are bridge calculi between λs and $\lambda\sigma$ and between λs_e and $\lambda\sigma$ and thus we are able to better understand one calculus in terms of the other. Finally, we present typed versions of all the calculi and check that the above mentioned isomorphism preserves types.

As a consequence, the $\lambda\omega$-calculus is a calculus in the $\lambda\sigma$-style that has the following properties a..g: a) $\lambda\omega$ simulates one step β-reduction, b) $\lambda\omega$ is confluent (on closed terms), c) $\lambda\omega$ preserves strong normalisation, d) $\lambda\omega$'s associated calculus of substitutions is SN, e) the simply typed $\lambda\omega$ calculus is SN, f) the $\lambda\omega$-calculus possesses an extension $\lambda\omega_e$ that is confluent on open terms (terms with eventual metavariables of sort `term` only), and g) the simply typed $\lambda\omega_e$ calculus is weakly normalising (on open term). As far as we know, the $\lambda\omega$-calculus is the first calculus in the $\lambda\sigma$-style that has all those properties a..g. However, the open problem of the SN of the associated calculus of substitution of $\lambda\omega_e$ remains unsolved and like in the case of $\lambda\sigma$, λv and λs_e, $\lambda\omega_e$ does not have PSN.

Keywords: λ-calculus, explicit substitutions, $\lambda\sigma$, λs.

Introduction

There is no better way to start than by quoting Abelson and Sussman in [AS86]: *Despite the fact that substitution is a "straightforward idea", it turns out to be sur-*

*Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, Scotland, *email*: fairouz@cee.hw.ac.uk

†Department of Computer Science, University of Buenos Aires, Pabellón I - Ciudad Universitaria (1428) Buenos Aires, Argentina, *email*:rios@dc.uba.ar

prisingly complicated to give a rigorous mathematical definition of the substitution process... Indeed, there is a long history of erroneous definitions of "substitution" in the literature of logic and programming semantics.

Most literature on the λ-calculus treats substitution as an atomic operation and leaves implicit the actual computational steps necessary to perform substitution. Substitution is usually defined with operators which do not belong to the language of the λ-calculus. In any real implementation, the substitution required by β-reduction (and similar higher-order operations) must be implemented via less complex operations. Thus, there is a conceptual gap between the theory of the λ-calculus and its implementation in programming languages and proof assistants. Explicit substitution attempts to bridge this gap without abandoning the setting of the λ-calculus.

By representing substitutions in the structure of terms and by providing (first-order) reductions to propagate the substitutions, explicit substitution provides a number of benefits. A major benefit is that explicit substitution allows more flexibility in ordering work. Propagating substitutions through a particular subterm can wait until the subterm is the focus of computation. This allows a choice among the substitutions to be performed, thus improving locality of reference. Obtaining more control over the ordering of work has become an important issue in functional programming language implementation (cf. [Pey87]). The flexibility provided by explicit substitution also allows postponing unneeded work indefinitely (i.e., avoiding it completely). This can yield profits, since implicit substitution can be an inefficient, maybe even exploding, process by the many repetitions it causes. Another benefit is that explicit substitution allows formal modeling of the techniques used in real implementations, e.g., environments. Because explicit substitution is closer to real implementations, it has the potential to provide a more accurate cost model. (This possibility is particularly interesting in light of the difficulty encountered in formulating a useful cost model in terms of graph reduction [LM96, Pey87].)

Proof assistants may benefit from explicit substitution, due to the desire to perform substitutions locally and in a formal manner. Local substitutions are needed as follows. Given $xx[x{:=}y]$, one may not be interested in having yy as the result of $xx[x{:=}y]$ but rather only $yx[x{:=}y]$. In other words, one only substitutes one occurrence of x by y and continues the substitution later. Theorem provers like Nuprl [Con86] and HOL [GM93] implement substitution which allows the local replacement of some abbreviated term. This avoids a size explosion when it is necessary to replace a variable by a huge term only in specific places to prove a certain theorem.

Formalisation helps in studying the termination and confluence properties of systems. Without formalisation, important properties such as the correctness of substitutions often remain un-established, causing mistrust in the implementation. In fact, it is known that the first implementation of substitution in Automath [NGdV94] was incorrect, and that most of the bugs in the implementation of LCF came from clashes of bound variables in strange situations [Pau90]. As the implementation of substitution in many theorem provers is not based on a formal system, it is not clear what properties their underlying substitution has, nor can their implementations be compared. Thus, it helps to have a choice of explicit substitution systems whose properties have already been established. This is witnessed by the recent theorem prover ALF, which is formally based on Martin-Löf's type theory with explicit substitution [Mag95]. Another justification for explicit substitution in theorem proving is that some researchers believe "tactics" can be replaced by the notion of incomplete proofs, which are believed to need explicit substitutions [Muñ97c, Muñ96, Muñ97a, Muñ98, Mag95]. Similarly, the area of implementations of functional and logic languages has witnessed an important research in explicit substitutions, e.g. [Ben97, NW90, DHK95].

The last fifteen years have seen an increasing interest in formalising substitution explicitly; various calculi including new operators to denote substitution have been proposed. Amongst these calculi we mention $C\lambda\xi\phi$ [dB78]; the calculi of categorical combinators [Cur86]; $\lambda\sigma$ [ACCL91], $\lambda\sigma_{\Uparrow}$ [CHL96], $\lambda\sigma_{SP}$ [Río93], referred to as the $\lambda\sigma$-family; $\lambda\upsilon$ [BBLRD96], the calculi of [FKP99] and $\lambda\zeta$ [Muñ97c], which are descendants of the $\lambda\sigma$-family; $\varphi\sigma BLT$ [KN93], $\lambda\chi$-calculus [LRD95], $\lambda\mathtt{x}$ [BR96], λs [KR95], λt [KR98], λs_e [KR97], and λl [Gui99b, Gui99a]. All these calculi (except $\lambda\mathtt{x}$) are described in a de Bruijn setting where natural numbers play the role of variables and the set of terms Λ on which substitution will be made explicit is defined by: $\Lambda ::= \mathbb{N} \mid (\Lambda\Lambda) \mid (\lambda\Lambda)$. But, why so many varieties of systems of explicit substitutions and the search still continues? The following section attempts to explain:

The $\lambda\sigma$-calculus (cf. [ACCL91]) reflects in its choice of operators and rules the calculus of categorical combinators (cf. [Cur86]). The main innovation of the $\lambda\sigma$-calculus is the division of terms in two sorts: sort `term` and sort `substitution`. Calculi à la λs depart from this style of explicit substitutions in two ways. First, they keep the classical and unique sort `term` of the λ-calculus. Second, they do not use some of the categorical operators, especially those which are not present in the classical λ-calculus. The main reason for doing so, is to remain closer to the λ-calculus from an intuitive point of view, rather than a categorical one.

But, what properties does one look for in calculi that are to be the basis for programming languages? We attempt to list some of those desired properties for calculi of explicit substitutions:

1. **Termination or Strong Normalisation (SN):** For a calculus of explicit substitutions $\lambda\mathtt{subst}$, does the underlying calculus of substitutions `subst` terminate? This question is of course important. One does not want to include non-terminating rules to the λ-calculus.

2. **Confluence (CR):** Is the substitution calculus $\lambda\mathtt{subst}$ confluent on:

   (a) Ground terms? (I.e. terms of Λ above with explicit substitutions)

   (b) Open terms? It is possible to consider, besides the classical variables (now numbers), real variables (which correspond to meta-variables in the classical setting). The terms obtained with this extended syntax are called *open terms* and they can be considered as *contexts*, the new variables corresponding to *place-holders*. The interest in studying the calculi on open terms is that they allow, for instance, the representation of incomplete proofs where the place-holder stands for the still unknown part of the proof. Calculi on open terms have also provided the tools to prune the search space in unification algorithms (cf. [DHK95]).

3. **Simulation of β-reduction:** If a evaluates in the λ-calculus (using only β-reduction) to b, does a evaluate to b in the λsubst-calculus (using the β-rule and the substitutions rules)?

4. **Preservation of Termination (PSN):** If a terminates in the λ-calculus, does it terminate in the λsubst-calculus?

$\lambda\sigma$ enjoyed properties 1, 2a, and 3 but not 2b. Therefore, $\lambda\sigma_{\Uparrow}$ [HL89, CHL96] was proposed. $\lambda\sigma_{\Uparrow}$ is a variant of $\lambda\sigma$ that is confluent on open terms. Nevertheless, 4 remained unknown for $\lambda\sigma$ or $\lambda\sigma_{\Uparrow}$ until Melliès proved that $\lambda\sigma_{\Uparrow}$ (as well as both the rest of the $\lambda\sigma$-family and the categorical combinators) does not preserve SN [Mel95]. This led to the creation of $\lambda\upsilon$ [Les94], $\lambda\mathtt{x}$ [BR96], λs [KR95] and λs_e [KR97] calculi. [BBLRD96] and[BR96] establish properties 1, 2a, 3 and 4 for $\lambda\upsilon$ and $\lambda\mathtt{x}$ but the first calculus has not been extended on open terms and the second has been extended on open terms but it is not clear which properties hold [LR98]. [KR97] establishes properties 2 and 3 for λs_e, but property 1 remains an open problem for λs_e and Guillaume [Gui99b, Gui99a] showed that PSN (property 4) fails for λs_e. Moreover, [Gui99b, Gui99a] proposes a calculus λl that has all the properties 1..4, but with a restricted form of 3. In this paper, we avoid any further discussion of the labelled calculus λl because it differs from calculi à la $\lambda\sigma$ and λs in that it uses labels and so relating it to the other styles is not straightforward.

$\lambda\sigma_{\Uparrow}$ satisfies 1, 2, and 3, whereas $\lambda\upsilon$ achieves 1, 2a, 3 and 4 by removing the composition operator. However, [FKP99] provides two calculi of explicit substitutions that have the composition operator and still have PSN. Remark that $\lambda\upsilon$ and the calculi of [FKP99] do not enjoy 2b.

The $\lambda\zeta$-calculus (cf. [Muñ97c]) has been proposed as a calculus which preserves strong normalisation and is itself confluent on open terms. In other words, $\lambda\zeta$ satisfies 1, 2, and 4. The $\lambda\zeta$-calculus works with two new applications that allow the passage of substitutions within classical applications only if these applications have a head variable. This is done to cut the branch of the critical pair which is responsible for the non-confluence of $\lambda\upsilon$ on open terms. Hence, $\lambda\zeta$ preserves strong normalisation and is itself confluent on open terms. Unfortunately, $\lambda\zeta$ is not able to simulate one step β-reduction as shown in [Muñ97c]. Instead, it simulates only a "big step" β-reduction. This is our reason for not discussing it further in this paper.

Another line of expliciting substitutions has been made in [KN93, KR95, KR97, KRW98]. In [KN93], the λ-calculus was rewritten using a notation influenced strongly by de Bruijn's notation for Automath [NGdV94]. In that notation [KN93], every λ-term is simply a sequence of *items* followed by a variable. This *item notation*, allowed also the introduction of so called *substitution items* and the inclusion of rules that explicit the passage of substitution. Alas however, the calculus of [KN93] does not satisfy 1 nor 2 nor 4. For this reason, [KR95] set out to find the part of the calculus of [KN93] that satisfies as much of 1 to 4 as possible. The solution was to extend the λ-calculus with explicit substitutions by turning de Bruijn's meta-operators into object-operators. (Mention of a very close calculus to the λs-calculus can be already found in [Cur86], exercise 1.2.7.2, where reference to previously unpublished notes of Y. Lafont is given.) The resulting calculus λs remains intuitively as close to the λ-calculus as possible for a calculus of explicit substitution. λs (like $\lambda\upsilon$) satisfies all of 1, 2a, 3 and 4. Moreover, λs has an extension λs_e (cf. [KR97]) that is confluent on open terms (hence λs_e satisfies 2a and 2b). Also, λs_e satisfies 3. It is still an open problem whether λs_e satisfies 1 and it has been established in [Gui99b, Gui99a] that it does not satisfy 4.

The presence of such varieties of calculi of explicit substitutions, makes it desirable to find a common framework between both styles so that maybe their complementary properties can be combined.

All the above discussion was concerned with the type-free λ-calculus extended with explicit substitutions. However, type theory is at the heart of the theory and implementation of programming languages and theorem provers. For this reason, no calculus can really bridge the gap between theory and implementation and be a useful one for programming languages and theorem proving if there was no way to accommodate types.

The results concerning typed calculi are the following. $\lambda\sigma$ does not preserve strong normalisation and the counterexample given in [Mel95] to prove it happens to be a very decent typable term. Therefore, typed $\lambda\sigma$ is not SN. On the other hand, $\lambda\upsilon$ preserves strong normalisation and its simply typed version is strongly normalising. The same applies to λs and λx which, (like $\lambda\upsilon$) preserve strong normalisation and have simply typed versions that are strongly normalising. [ACCL91] had typed versions of $\lambda\sigma$ but only recently, $\lambda\sigma$ (with open terms) has been shown to be weakly normalising [GL97]. Extending second and higher order λ-calculus with explicit substitutions remains an active subject of research[Bon99, Blo99, Blo97, Muñ97b].

We believe that a comparison between the two styles and a formulation of λs and λs_e in the $\lambda\sigma$-style could be useful to better understand one calculus in terms of the other. Therefore, we start by focusing on $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\upsilon$, λs, λt and λu. All these calculi are rewriting systems on a set of terms that contain the classical terms of the λ-calculus (*pure terms*). All of them possess a rule to start β-reduction (the only rule of the λ-calculus) and a set of rules to compute the substitution generated by this starting rule.

Since calculi with explicit substitutions are intended to extend the classical λ-calculus, it is expected that β-reduction could be recovered in some way within these calculi, for instance, if $\lambda\xi$ is an explicit substitution calculus, we may have for pure terms a, b:

1. *one step simulation*: if $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda\xi} b$.
2. *big step simulation*: if $a \twoheadrightarrow_\beta b$ and b is in β-normal form then $a \twoheadrightarrow_{\lambda\xi} b$.

The calculi $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\upsilon$, λs, λt, λu have the property of *one step simulation* and we concentrate in this paper on the adequacy of this simulation which implies the big step one, leaving the study of the adequacy of the latter for future work. Our criterion of adequacy is essentially the following: we say that the calculus $\lambda\xi_1$ is more adequate than the calculus $\lambda\xi_2$ if for every simulation of a classical β-step in $\lambda\xi_2$ there is a shorter simulation in $\lambda\xi_1$.

There are reasons why we do not consider the other calculi in our study of adequacy as defined in this paper. For example, $\lambda\zeta$ (the only calculus that) simulates just a big step β-reduction (and hence it does not make sense to study its adequacy in our sense), whereas λs_e, $\varphi\sigma BLT$ and $\lambda\sigma_{SP}$ are less interesting because they are less well-behaved calculi of explicit substitutions.

In section 1 we introduce the formal machinery, recall the various calculi and their properties, present the λu-calculus and give the formal statement of the criterion of adequacy to simulate β-reduction.

In section 2 we use our criterion to compare several of the above mentioned calculi. We conclude that λt is more adequate than $\lambda\upsilon$, and that λu is more adequate than λs, $\lambda\upsilon$ and $\lambda\sigma_{\Uparrow}$.

In section 3 we give counterexamples to show the calculi that are incomparable according to our criterion, namely: λt cannot be compared with λu, λs, $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$; λu cannot be compared with $\lambda\sigma$ and λt; λs cannot be compared with λt, $\lambda\upsilon$, $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$. We show also that, surprisingly, no comparison is possible between any two calculi in the $\lambda\sigma$-style.

In Section 4, we provide the $\lambda\omega$ and $\lambda\omega_e$ calculi, which are two-sorted: sort `term` and sort `substitution`, and hence closer to $\lambda\sigma$. When restricting these calculi to the sort `term` we obtain calculi which are isomorphic to λs and λs_e, respectively.

In Section 5 we give the isomorphisms between $\lambda\omega$ and λs and between $\lambda\omega_e$ and λs_e which enable us to establish that $\lambda\omega$ (resp. $\lambda\omega_e$) has the same properties of λs (resp. λs_e).

In Section 6 we recall the typed versions of λs, λs_e and $\lambda\sigma$ and introduce the typed $\lambda\omega$ and $\lambda\omega_e$ calculi. We prove that the isomorphism introduced in Section 5 preserves types and we conclude by establishing Subject Reduction for our calculi.

1 Preliminaries

We assume the reader familiar with de Bruijn indices (cf. [dB72]) and with notions of reduction as in [Bar84]. In particular, $a = b$ is used to mean that a and b are syntactically identical; and for a reduction notion R, we denote with $\stackrel{=}{\to}_R$ the reflexive closure of R, with $\twoheadrightarrow_R$ or just $\twoheadrightarrow$ the reflexive and transitive closure of R and with $\twoheadrightarrow_R^+$ or just $\twoheadrightarrow^+$ or just $\to^+$ 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. The following is needed:

Definition 1 Let R be a reduction on A. We define (local) confluence or (W)CR ((weakly) Church Rosser), normal forms and normalisation as follows:

1. R is WCR when $\forall a, b, c \in A\ \exists d \in A\ ((a \to b \wedge a \to c) \Rightarrow (b \twoheadrightarrow d \wedge c \twoheadrightarrow d))$.
2. R is CR when $\forall a, b, c \in A\ \exists d \in A\ ((a \twoheadrightarrow b \wedge a \twoheadrightarrow c) \Rightarrow (b \twoheadrightarrow d \wedge c \twoheadrightarrow d))$.
3. $a \in A$ is an R-normal form (R-nf for short) if there is no $b \in A$ such that $a \to b$.
4. b has a normal form if there exists a nf a such that $b \twoheadrightarrow a$.
5. R is strongly normalising (SN) if there is no infinite sequence $(a_i)_{i \geq 0}$ where $\forall\, i \geq 0$, $a_i \to a_{i+1}$.

Note that 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 $R(a)$.

1.1 The classical λ-calculus in de Bruijn notation

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 $m, n, \ldots$ to range over $\mathbb{N}$ (positive natural numbers). Furthermore, we assume the usual conventions about parentheses and avoid them when no confusion occurs. We say that a reduction $\rightarrow$ is *compatible on* Λ when for all a, b, $c \in \Lambda$, we have $a \rightarrow b$ implies $a\,c \rightarrow b\,c$, $c\,a \rightarrow c\,b$ and $\lambda a \rightarrow \lambda b$.

In order to define β-reduction à la de Bruijn, we must define the substitution of a variable $\mathtt{n}$ for a term b in a term a. Therefore, we need to update the term b:

Definition 2 The *updating functions* $U_k^i : \Lambda \rightarrow \Lambda$ for $k \geq 0$ and $i \geq 1$ are defined inductively:

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

$$U_k^i(\lambda a) = \lambda(U_{k+1}^i(a))$$

Now we define the family of meta-substitution functions:

Definition 3 The *meta-substitution at level* j, for $j \geq 1$, of a term $b \in \Lambda$ in a term $a \in \Lambda$, denoted $a\{\!\{\mathtt{j} \leftarrow b\}\!\}$, is defined inductively on a as follows:

$$(a_1 a_2)\{\!\{\mathtt{j} \leftarrow b\}\!\} = (a_1\{\!\{\mathtt{j} \leftarrow b\}\!\})\,(a_2\{\!\{\mathtt{j} \leftarrow b\}\!\}) \qquad \mathtt{n}\{\!\{\mathtt{j} \leftarrow b\}\!\} = \begin{cases} \mathtt{n - 1} & \text{if } n > j \\ U_0^j(b) & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases}$$

$$(\lambda a)\{\!\{\mathtt{j} \leftarrow b\}\!\} = \lambda(a\{\!\{\mathtt{j+1} \leftarrow b\}\!\})$$

The following gives the properties of meta-substitution and updating (cf. [KR95]):

Lemma 4 Let a, b, $c \in \Lambda$. We have:

1. for $k < n < k + i$: $U_k^{i-1}(a) = U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$.
2. for $l \leq k < l + j$: $U_k^i(U_l^j(a)) = U_l^{j+i-1}(a)$.
3. for $k + i \leq n$: $U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n - i + 1} \leftarrow b\}\!\})$.
4. for $i \leq n$: $a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n + 1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n - i + 1} \leftarrow c\}\!\}\}\!\}$.
5. for $l + j \leq k + 1$: $U_k^i(U_l^j(a)) = U_l^j(U_{k+1-j}^i(a))$.
6. for $n \leq k + 1$: $U_k^i(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U_{k+1}^i(a)\{\!\{\mathtt{n} \leftarrow U_{k-n+1}^i(b)\}\!\}$.

Definition 5 *β-reduction* is the least compatible reduction on Λ generated by:

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

The *λ-calculus (à la de Bruijn)*, is the reduction system whose only rewriting rule is β.

Theorem 6 The λ-calculus à la de Bruijn is confluent.

(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)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$

Figure 1: The $\lambda\sigma$-rules

1.2 Calculi à la $\lambda\sigma$

In this section, we introduce the $\lambda\xi$-calculi (for $\xi \in \{\sigma, \sigma_{DB}, \sigma_{\Uparrow}, \upsilon\}$) which work on 2-sorted terms: *(proper) terms* and *substitutions*. The $\lambda\sigma$-calculus was introduced in [ACCL91] and the version presented there uses only the de Bruijn index 1 and the other de Bruijn indices are coded. We introduce here another version, denoted $\lambda\sigma_{DB}$, which uses all the de Bruijn indices and hence is at the same level with the other calculi studied in this paper. We introduce $\lambda\sigma_{DB}$ because it could be argued that the coding of the de Bruijn indices could change the status of $\lambda\sigma$ with respect to adequacy results. However, we show that $\lambda\sigma$ and $\lambda\sigma_{DB}$ have the same behaviour as far as comparison of adequacy with the other calculi studied here is concerned.

For every ξ, we use $a, b, c, \ldots$ to range over the set of terms $\Lambda\xi^t$, and $s, t, \ldots$ to range over the set of substitutions $\Lambda\xi^s$. We use $\lambda\xi$ to denote the set of rules of the $\lambda\xi$-calculus (which contains a rule *(Beta)*) and take the ξ-calculus to be the calculus whose rules are $\lambda\xi - \{(Beta)\}$. The $\lambda\xi$-calculus is the reduction system $(\Lambda\xi, \rightarrow_{\lambda\xi})$, where $\rightarrow_{\lambda\xi}$ is the least compatible (with the corresponding operators) reduction on $\Lambda\xi$ generated by the set of rules $\lambda\xi$.

For every $\xi \in \{\sigma, \sigma_{\Uparrow}, \upsilon\}$ (see [ACCL91, CHL96, BBLRD96]), the ξ-calculus is SN and the $\lambda\xi$-calculus is confluent on closed terms. Moreover, only the $\lambda\sigma_{\Uparrow}$-calculus is confluent on open terms (terms with variables of sort *term* and *substitution*) and only the $\lambda\upsilon$-calculus satisfies Preservation of Strong Normalisation (PSN) (all the calculi in the $\lambda\sigma$-family were shown in [Mel95] not to possess PSN; the $\lambda\upsilon$-calculus removes the composition of substitutions to guarantee PSN).

Definition 7 (The $\lambda\sigma$-calculus) Terms and substitutions of the *$\lambda\sigma$-calculus* are given by: $\Lambda\sigma^t ::= \mathtt{1} \mid \Lambda\sigma^t\Lambda\sigma^t \mid \lambda\Lambda\sigma^t \mid \Lambda\sigma^t[\Lambda\sigma^s]$ and $\Lambda\sigma^s ::= id \mid \uparrow \mid \Lambda\sigma^t \cdot \Lambda\sigma^s \mid \Lambda\sigma^s \circ \Lambda\sigma^s$. The set of rules $\lambda\sigma$ is given in Figure 1.

For every substitution s we define the *iteration of the composition of s* inductively

(Id)	$a[id]$	$\longrightarrow$	a
(IdR)	$s \circ id$	$\longrightarrow$	s
(VarShift)	$1 \cdot \uparrow$	$\longrightarrow$	id
(SCons)	$1[s] \cdot (\uparrow \circ s)$	$\longrightarrow$	s

Figure 2: The rules added to $\lambda\sigma$ to get $\lambda\sigma_{SP}$

as $s^1 = s$ and $s^{n+1} = s \circ s^n$. We use the convention $s^0 = id$. Note that the only de Bruijn index used is 1, but we can code $\mathtt{n}$ by the term $1[\uparrow^{n-1}]$. By so doing, we have $\Lambda \subset \Lambda\sigma^t$.

β-reduction of the λ-calculus is interpreted in the $\lambda\sigma$-calculus in two steps. The first, obtained by the application of *(Beta)*, consists in generating the substitution. The second step executes the propagation of this substitution, using the set of the σ-rules, until the concerned variables are reached. The reader is invited to check that $(\lambda\lambda 521)(\lambda 31) \twoheadrightarrow_{\lambda\sigma} \lambda 4(\lambda 41)1$.

It is well known that the $\lambda\sigma$-calculus is not confluent on open terms, furthermore it is not even locally confluent. To obtain local confluence four rules must be added, and the calculus thus obtained is called the $\lambda\sigma_{SP}$-calculus.

Definition 8 The $\lambda\sigma_{SP}$-calculus is obtained by adding to $\lambda\sigma$ the rules given in Figure 2 and by deleting the rules *(VarId)* and *(ShiftId)*, since both of them are instances of the new rules.

Even the $\lambda\sigma_{SP}$-calculus is not confluent on open terms (terms which admit metavariables of both sorts), as shown in [CHL96], but it is confluent when the set of open terms is restricted to those which admit metavariables of sort `term` only [Río93].

Definition 9 (The $\lambda\sigma_{DB}$-calculus) The syntax of the $\lambda\sigma_{DB}$-calculus is exactly that of the $\lambda\sigma$-calculus except that 1 is replaced by $\mathbb{N}$. The set, $\lambda\sigma_{DB}$, of rules of the $\lambda\sigma_{DB}$*-calculus* is $\lambda\sigma$ where *(VarId)* is replaced by $a[id] \to a$ plus the three extra rules: $\mathtt{n}+1[a \cdot s] \to \mathtt{n}[s]$, $\mathtt{n}[\uparrow] \to \mathtt{n}+1$ and $\mathtt{n}[\uparrow \circ s] \to \mathtt{n}+1[s]$.

Definition 10 (The $\lambda\upsilon$-calculus) Terms and substitutions of the $\lambda\upsilon$-calculus are given by: $\Lambda\upsilon^t ::= \mathbb{N} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s]$ and $\Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t$. For $a \in \Lambda\upsilon^t$, $s \in \Lambda\upsilon^s$, $\Uparrow^n(s)$ is given by: $\Uparrow^0(s) = s$, $\Uparrow^{n+1}(s) = \Uparrow(\Uparrow^n(s))$ and $a[s]^i$ by: $a[s]^0 = a$, $a[s]^{n+1} = (a[s]^n)[s]$. The set of rules $\lambda\upsilon$ is given in Figure 3.

Definition 11 (The $\lambda\sigma_{\Uparrow}$-calculus) Terms and substitutions of the $\lambda\sigma_{\Uparrow}$-calculus are given by:

$\Lambda\sigma^t_{\Uparrow} ::= \mathbb{N} \mid \Lambda\sigma^t_{\Uparrow}\Lambda\sigma^t_{\Uparrow} \mid \lambda\Lambda\sigma^t_{\Uparrow} \mid \Lambda\sigma^t_{\Uparrow}[\Lambda\sigma^s_{\Uparrow}]$
$\Lambda\sigma^s_{\Uparrow} ::= id \mid \uparrow \mid \Uparrow(\Lambda\sigma^s_{\Uparrow}) \mid \Lambda\sigma^t_{\Uparrow} \cdot \Lambda\sigma^s_{\Uparrow} \mid \Lambda\sigma^s_{\Uparrow} \circ \Lambda\sigma^s_{\Uparrow}$.

For $s \in \Lambda\sigma^s_{\Uparrow}$, s^n is given by: $s^1 = s$, $s^{n+1} = s \circ s^n$ and as in Definition 10, we define $\Uparrow^n(s)$ by: $\Uparrow^0(s) = s$, $\Uparrow^{n+1}(s) = \Uparrow(\Uparrow^n(s))$.
The set of rules $\lambda\sigma_{\Uparrow}$ is given in Figure 4.

(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)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$

Figure 3: The $\lambda\upsilon$-rules

1.3 Calculi à la λs

Calculi à la λs avoid introducing two different sets of entities and insist on remaining close to the syntax of the λ-calculus using de Bruijn indices[1]. Next to λ and application, they introduce substitution (σ, ς) and updating (φ, θ) operators. We shall introduce three such calculi: λs, λt and λu. We let a, b, c, etc. range over the sets of terms Λs, Λt and Λu. A term containing neither substitution nor updating operators is called a *pure term*. For $\xi \in \{s, t, u\}$, the $\lambda\xi$- and ξ-calculi are defined as in the previous section (take σ- or ς-*generation* instead of *Beta*) from a set of rules $\lambda\xi$ or ξ.

The λs-calculus was introduced in [KR95] with the aim of providing a calculus that preserves strong normalisation and has a confluent extension on open terms [KR97]. The λt-calculus is a variant of λs that updates partially, as the $\lambda\sigma$-calculi do. The λu-calculus is introduced here for the first time and is only a slight (yet more adequate) variation of λs. In [KR95, KR98], we establish the properties of these calculi which we list in the following theorem.

Theorem 12 For $\xi \in \{s, t, u\}$, the ξ-calculus is SN, the $\lambda\xi$-calculus is confluent on closed terms and satisfies PSN. Moreover, the $\lambda\xi$-calculus for $\xi \in \{s, u\}$ simulates β-reduction, is sound and has a confluent extension on open terms.

Definition 13 (The λs-calculus) Terms of the λs-calculus are given by:

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

The set of rules λs is given in Figure 5.

Definition 14 (The λt-calculus) Terms of the λt-calculus are given by:

[1] It can be argued that because we use de Bruijn indices, we remain close to de Bruijn's philosophy rather than to the syntax of the λ-calculus and that instead it is calculi like λx of [BR96] and $\lambda\chi$ of [LRD95] that remain close to the syntax of the lambda calculus. So, we need to explain here that by staying with the syntax of the λ-calculus we mean that we do not introduce substitutions and other categorical operators separately as in $\lambda\sigma$, but that a term for us is either an abstraction term, an application term, a substitution term or an updating term.

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b \cdot id]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(Varshift1)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$
(Varshift2)	$\mathtt{n}\,[\uparrow \circ s]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}\,[s]$
(FVarCons)	$\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	a
(RVarCons)	$\mathtt{n}+\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	$\mathtt{n}\,[s]$
(FVarLift1)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(FVarLift2)	$\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{1}\,[t]$
(RVarLift1)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}[s \circ \uparrow]$
(RVarLift2)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{n}[s \circ (\uparrow \circ t)]$
(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(Ass)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\longrightarrow$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(LiftId)	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a\,[id]$	$\longrightarrow$	a

Figure 4: The $\lambda\sigma_{\Uparrow}$-rules

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	$\mathtt{n}\, \sigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if } n > i \\ \varphi_0^i\, b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i \end{cases}$
φ-λ-transition	$\varphi_k^i(\lambda a)$	$\longrightarrow$	$\lambda(\varphi_{k+1}^i\, a)$
φ-app-transition	$\varphi_k^i(a_1\, a_2)$	$\longrightarrow$	$(\varphi_k^i\, a_1)\, (\varphi_k^i\, a_2)$
φ-destruction	$\varphi_k^i\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$

Figure 5: The λs-rules

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

For $a \in \Lambda t$, we define $\theta_k^0 a = a$ and $\theta_k^{i+1}(a) = \theta_k(\theta_k^i(a))$. The set of rules λt is given in Figure 6.

The main difference between λt and λs can be summarised 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 $\lambda\sigma$-caculi, $\lambda\upsilon$ and $\lambda\zeta$ since all of them introduce an updating operator in their *(Abs)*-rule.

We introduce now an adequate variation on λs where in the *σ-destruction* rule, the case $n = i = 1$ is treated in a more adequate way which does not introduce the operator φ_0^1 since the computation $\varphi_0^1(b)$ will finally evaluate to b.

Definition 15 (The λu-calculus) Terms of the λu-calculus are given by:

$$\Lambda u ::= \mathbb{N} \mid \Lambda u \Lambda u \mid \lambda \Lambda u \mid \Lambda u\, \sigma^j \Lambda u \mid \varphi_k^i \Lambda u \quad \textit{where} \;\; i \geq 2,\; j \geq 1,\; k \geq 0\,.$$

and the set of rules λu is given in Figure 7.

1.4 The λs_e-calculus

We introduce the open terms and the rules that extend λs to obtain the λs_e-calculus.

Definition 16 The set of *open terms*, noted Λs_{op} is given as follows:

$$\Lambda s_{op} ::= \mathbf{V} \mid \mathbb{N} \mid \Lambda s_{op} \Lambda s_{op} \mid \lambda \Lambda s_{op} \mid \Lambda s_{op}\, \sigma^j \Lambda s_{op} \mid \varphi_k^i \Lambda s_{op} \quad \textit{where} \;\; j, i \geq 1\,, \;\; k \geq 0$$

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

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	$\mathtt{n}\, \varsigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if } n > i \\ b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i \end{cases}$
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	$\theta_k\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$

Figure 6: The λt-rules

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	$\mathtt{n}\, \sigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if } n > i \\ \varphi^i_0\, b & \text{if } n = i > 1 \\ b & \text{if } n = i = 1 \\ \mathtt{n} & \text{if } n < i \end{cases}$
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	$\varphi^i_k\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$

Figure 7: The λu-rules

σ-σ-transition	$(a\,\sigma^i b)\,\sigma^j\,c$	$\longrightarrow$	$(a\,\sigma^{j+1}\,c)\;\sigma^i\;(b\,\sigma^{j-i+1}\,c)$	if	$i \leq j$
σ-φ-transition 1	$(\varphi^i_k\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi^{i-1}_k\,a$	if	$k < j < k+i$
σ-φ-transition 2	$(\varphi^i_k\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi^i_k(a\,\sigma^{j-i+1}\,b)$	if	$k+i \leq j$
φ-σ-transition	$\varphi^i_k(a\,\sigma^j\,b)$	$\longrightarrow$	$(\varphi^i_{k+1}\,a)\,\sigma^j\;(\varphi^i_{k+1-j}\,b)$	if	$j \leq k+1$
φ-φ-transition 1	$\varphi^i_k\;(\varphi^j_l\;a)$	$\longrightarrow$	$\varphi^j_l\;(\varphi^i_{k+1-j}\,a)$	if	$l+j \leq k$
φ-φ-transition 2	$\varphi^i_k\;(\varphi^j_l\;a)$	$\longrightarrow$	$\varphi^{j+i-1}_l\;a$	if	$l \leq k < l+j$

Figure 8: The new rules of the λs_e-calculus

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

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

and $(X\sigma^1 Y)\sigma^1 \mathtt{1}$ and $((\lambda X)\sigma^1 \mathtt{1})(Y\sigma^1 \mathtt{1})$ have no common reduct. Moreover, the above example shows that even local confluence is lost. But since $((\lambda X)\sigma^1 \mathtt{1})(Y\sigma^1 \mathtt{1}) \twoheadrightarrow (X\sigma^2 \mathtt{1})\sigma^1(Y\sigma^1 \mathtt{1})$, the solution to the problem seems at hand if one has in mind the properties of meta-substitutions and updating functions of the λ-calculus in the Bruijn notation (cf. Lemma 4). These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to λs yield a rewriting system which happens to be locally confluent. For instance, the rule corresponding to the Meta-substitution lemma (Lemma 4.4) is the *σ-σ-transition* rule. The addition of this rule solves the critical pair in our counterexample, since now we have $(X\sigma^1 Y)\sigma^1 \mathtt{1} \to (X\sigma^2 \mathtt{1})\sigma^1(Y\sigma^1 \mathtt{1})$.

Definition 17 The set of rules λs_e is obtained by adding the rules given in Figure 8 to the set λs. The *λs_e-calculus* is the reduction system $(\Lambda s_{op}, \to_{\lambda s_e})$ where $\to_{\lambda s_e}$ is the least compatible reduction on Λs_{op} generated by the set of rules λs_e. The *calculus of substitutions associated with the λs_e-calculus* is the rewriting system generated by the set of rules $s_e = \lambda s_e - \{\sigma\text{-}generation\}$ and we call it *s_e-calculus.*

In [KR97] we proved the following:

Theorem 18 (WN and CR of s_e) The s_e-calculus is weakly normalising and confluent.

Lemma 19 (Simulation of β-reduction) Let $a,\, b \in \Lambda$, if $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda s_e} b$.

Theorem 20 (CR of λs_e) The λs_e-calculus is confluent on open terms.

Theorem 21 (Soundness) Let $a,\, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda s_e} b$ then $a \twoheadrightarrow_\beta b$.

1.5 The criterion of adequacy

We give now a formal presentation of the criterion of adequacy we use to compare the different calculi.

Definition 22 Let $a, b \in \Lambda$ such that $a \to_\beta b$. A *simulation* of this β-reduction in $\lambda\xi$ for $\xi \in \{\sigma, \sigma_\Uparrow, \upsilon, s, t, u\}$ is a $\lambda\xi$-derivation $a \to_r c \twoheadrightarrow_\xi \xi(c) = b$ where r is the rule starting β (*(Beta)* for the calculi in the $\lambda\sigma$-style and σ- or ς-*generation* for the calculi in the λs-style) applied to the same redex as the redex in $a \to_\beta b$. We say that the $\lambda\xi$-calculus *simulates β-reduction* if every β-reduction $a \to_\beta b$ has a simulation in $\lambda\xi$.

The following was shown for each of the calculi we consider (see the relevant articles):

Lemma 23 For $\xi \in \{\sigma, \sigma_\Uparrow, \upsilon, s, t, u\}$, $\lambda\xi$ simulates β-reduction.

Definition 24 Let $\xi_1, \xi_2 \in \{\sigma, \sigma_\Uparrow, \upsilon, s, t, u\}$. The $\lambda\xi_1$-calculus is *more adequate (in simulating one step β-reductions) than* the $\lambda\xi_2$-calculus, denoted $\lambda\xi_1 \prec \lambda\xi_2$, if

1. for every classical β-reduction $a \to_\beta b$ and every $\lambda\xi_2$-simulation $a \twoheadrightarrow^n_{\lambda\xi_2} b$ there exists a $\lambda\xi_1$-simulation $a \twoheadrightarrow^m_{\lambda\xi_1} b$ such that $m \leq n$.
2. there exist a classical β-reduction $a \to_\beta b$ and a $\lambda\xi_1$-simulation $a \twoheadrightarrow^m_{\lambda\xi_1} b$ such that for every $\lambda\xi_2$-simulation $a \twoheadrightarrow^n_{\lambda\xi_2} b$ we have $m < n$.

It is easy to verify that $\prec$ is transitive and asymmetric.

2 Establishing adequacy

In this section we put the criterion at work. The main idea is to define functions (denoted with Q) which evaluate the length of the derivations of certain families of terms that contain the contracta of the *(Beta)*- rules (eg. $a[b/]$ in $\lambda\upsilon$). For $\lambda\upsilon$ it is possible to prove that all these derivations have the same length, whereas for $\lambda\sigma_\Uparrow$ our functions compute just the length of the shortest derivation. To define these Q-functions we need to define another functions (denoted with M) which evaluate the length of the derivations of updatings. For the scope of this section, only the M-functions are needed for λt and λu.

2.1 λt is more adequate than $\lambda\upsilon$

We introduce a set of terms $\Lambda_\theta \subset \Lambda t$ on which induction will be used to define M^t (a function that computes the length of derivations of updatings in λt). 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 25 $\Lambda_\theta ::= \mathbb{N} \,|\, \Lambda_\theta\Lambda_\theta \,|\, \lambda\Lambda_\theta \,|\, \theta_k\Lambda_\theta$, where $k \geq 0$. The *length* of terms in Λ_θ is defined by: $L_\theta(\mathtt{n}) = 1$, $L_\theta(ab) = L_\theta(a) + L_\theta(b) + 1$, $L_\theta(\lambda a) = L_\theta(\theta_k a) = L_\theta(a) + 1$. By induction on $a \in \Lambda_\theta$ we mean induction on $L_\theta(a)$.

Remark 26 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, $L_\theta(a) \geq L_\theta(t(\theta_k a))$.

Remark 27 It is easy to show by induction on a that if $a \in \Lambda_\theta$ and $a \to_t b$ then $b \in \Lambda_\theta$.

Definition 28 We define $M^t : \Lambda_\theta \to \mathbb{N}$ by induction as follows:

$M^t(\mathtt{n}) = 1$
$M^t(ab) = M^t(a) + M^t(b) + 1$
$M^t(\lambda a) = M^t(a) + 1$
$M^t(\theta_k a) = M^t(t(\theta_k a)) + M^t(a)$

Remark that the previous definition is correct thanks to Remark 26: $M^t(\theta_k a)$ can be inductively defined in terms of $M^t(t(\theta_k a))$ because $L_\theta(t(\theta_k a)) \leq L_\theta(a) < L\theta(\theta_k(a))$.

Lemma 29 For $a \in \Lambda_\theta$, every t-derivation of $\theta_k a$ to its t-normal form has length $M^t(a)$.

Proof It is immediate to show that $\to_t$ has the diamond property on Λ_θ, i.e. for $a \in \Lambda\theta$, if $a \to_t b$ and $a \to_t c$ then either $b = c$ or there exists d such that $b \to_t d$ and $c \to_t d$. Therefore it is easy to conclude that all the derivations of a term to its normal form have the same length.

Now we show that any derivation of $\theta_k(a)$ to its normal form has length $M^t(a)$, by induction on a and analyzing just one derivation.

- If $a = \mathtt{m}$ it is obvious.
- If $a = bc$ we conclude by reducing at the root and applying I.H..
- If $a = \lambda b$ we conclude as in the previous case.
- If $a = \theta_k(\theta_m(b))$ we first reduce $\theta_m(b)$ to its normal form $t(\theta_m(a)$ in $M^t(a_1)$ steps by I.H. and then, again by I.H. (which can be applied because of Remark 26) we take $\theta_k(t(\theta_m(a)))$ into its normal form in $M^t(t(\theta_n(a))$.

Corollary 30 For $a \in \Lambda_\theta$, all the t-derivations of $\theta_k^i a$ to its t-normal form have the same length, namely $(i-1)M^t(t(a)) + M^t(a)$.

Proof Prove first by induction on $a \in \Lambda_\theta$, using Remark 26, that $M^t(t(a)) = M^t(t(\theta_k a))$, then use this result to prove, by induction on $j \geq 1$, that $M^t(t(a)) = M^t(t(\theta_k^j a))$. Use now Definition 28 and the two previous results to show, by induction on $l \geq 1$, that $M^t(\theta_k^l(a)) = lM^t(t(a)) + M^t(a)$. Finally, use Lemma 29 and the last result with $l = i - 1$ to prove the corollary. Note that the hypothesis $a \in \Lambda_\theta$ (and hence Definition 25) are essential.

Now we are going to prove the corresponding results for $\lambda\upsilon$.

Definition 31 $\Lambda_\uparrow ::= \mathbb{N} \mid \Lambda_\uparrow\Lambda_\uparrow \mid \lambda\Lambda_\uparrow \mid \Lambda_\uparrow[\Uparrow^k(\uparrow)]$, 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 32 Let $a \in \Lambda_\uparrow$ and $k \geq 0$, then $L_\uparrow(a) \geq L_\uparrow(\upsilon(a[\Uparrow^k(\uparrow)]))$.

Remark 33 If $a \in \Lambda_\uparrow$ and $a \to_\upsilon b$ then $b \in \Lambda_\uparrow$.

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

$$M_k^v(\mathtt{n}) = \begin{cases} 2k+1 & \text{if } n > k \\ 2n-1 & \text{if } n \leq k \end{cases}$$

$$M_k^v(ab) = M_k^v(a) + M_k^v(b) + 1$$

$$M_k^v(\lambda a) = M_{k+1}^v(a) + 1$$

$$M_k^v(a[\Uparrow^p(\uparrow)]) = M_k^v(v(a[\Uparrow^p(\uparrow)])) + M_p^v(a)$$

Lemma 35 For $a \in \Lambda_\uparrow$, all the v-derivations of $a[\Uparrow^k(\uparrow)]$ to its v-nf have length $M_k^v(a)$.

Proof Induction (on the weight used in [BBLRD96] to show SN for v) and case analysis.

Corollary 36 For $a \in \Lambda_\uparrow$, all the v-derivations of $a[\Uparrow^k(\uparrow)]^i$ to its v-normal form have the same length, namely $(i-1)M_k^v(v(a)) + M_k^v(a)$.

Lemma 37 Let $b \in \Lambda$, for every derivation $b[\Uparrow^k(\uparrow)]^i \twoheadrightarrow_v^m v(b[\Uparrow^k(\uparrow)]^i)$ there exists $n \leq m$ such that $\theta_p^i b \twoheadrightarrow_t^n t(\theta_p^i b)$.

Proof Prove first that for every $b \in \Lambda$ and $k \geq 0$, $M_k(b) \geq M(b)$ by induction on $b \in \Lambda$. Conclude using lemmas 29 and 35.

Definition 38 Let $a, b \in \Lambda$ and $i \geq 0$, we define $Q_i^v(a,b)$ by induction on a:

$$Q_i^v(\mathtt{n}, b) = \begin{cases} 2i+1 & \text{if } n > i+1 \\ 2n-1 & \text{if } n < i+1 \\ i(1+M_0^v(b))+1 & \text{if } n = i+1 \end{cases}$$

$$Q_i^v(cd, b) = Q_i^v(c,b) + Q_i^v(d,b) + 1$$

$$Q_i^v(\lambda c, b) = Q_{i+1}^v(c,b) + 1$$

Lemma 39 Let $a, b \in \Lambda$ and $i \geq 0$, all the v-derivations of $a[\Uparrow^i(b/)]$ to its v-nf have the same length, namely $Q_i^v(a,b)$.

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 36.

Lemma 40 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 $Q_i^v(a,b)$.

Proof By induction on a reducing always at the root. For the case $a = \mathtt{i+1}$ use the fact that $M_0^v(b) \geq M^t(b)$ (induction on $b \in \Lambda$) and Corollary 30.

Theorem 41 λt is more adequate than λv.

Proof Show by induction on a that for $a \in \Lambda$, and a λv-derivation $a \to_B b \twoheadrightarrow_v^m v(b)$, there exists $n \leq m$ where $a \to_{\varsigma-gen} c \twoheadrightarrow_t^n t(c)$.
The interesting case is $a = (\lambda d)e \to_B d[e/] \twoheadrightarrow^m v(d[e/])$. By Lemmas 39 and 40, $m = Q_0^v(d,e)$ and there exists a derivation $d\varsigma^1 e \twoheadrightarrow_t^n t(d\varsigma^1 e)$ such that $n \leq Q_0^v(d,e)$.

To check the second condition in Definition 24 remark that there are an infinity of cases for which the inequality is strict. For instance, take $(\lambda\lambda\ldots\lambda\mathtt{n})a$ with m λ's and $n > m > 1$. It is easy to check, using the function Q^v_{m-1} that $3m-2$ reductions are needed to simulate β-reduction in λv, whereas only $m+1$ reductions are sufficient in λt. Also, for $m > n$ the number of reductions needed in λv is also strictly greater than the number needed in λt.

2.2 λu is more adequate than $\lambda\sigma_{\Uparrow}$

Definition 42 For $k \geq 0$ and $i \geq 1$, we define $M^{\Uparrow}_{ki} : \Lambda \to \mathbb{N}$ by induction as follows:

$$M^{\Uparrow}_{ki}(\mathtt{n}) = \begin{cases} 2n-1 & \text{if } n < k+1 \\ 2(k+i)-1 & \text{if } n \geq k+1 \end{cases}$$

$$M^{\Uparrow}_{ki}(ab) = M^{\Uparrow}_{ki}(a) + M^{\Uparrow}_{ki}(b) + 1$$

$$M^{\Uparrow}_{ki}(\lambda a) = M^{\Uparrow}_{k+1\ i}(a) + 1$$

Lemma 43 For $a \in \Lambda$, every $\sigma_{\Uparrow}$-derivation of $a[\Uparrow^k\ (\uparrow^i)]$ to its $\sigma_{\Uparrow}$-nf has length $M^{\Uparrow}_{ki}(a)$.

Proof By induction on a, controlling all the possible $\sigma_{\Uparrow}$-derivations.

Definition 44 For $k \geq 0$ and $i \geq 1$, we define $Q^{\Uparrow}_k : \Lambda \times \Lambda \to \mathbb{N}$ by induction as follows:

$$Q^{\Uparrow}_k(\mathtt{n}, c) = \begin{cases} 2n-1 & \text{if } n < k+1 \\ M^{\Uparrow}_{0\ n-1}(c)+n+1 & \text{if } n = k+1,\ k > 0 \\ 1 & \text{if } n = 1,\ k = 0 \\ 2k+3 & \text{if } n > k+1 \end{cases}$$

$$Q^{\Uparrow}_k(ab, c) = Q^{\Uparrow}_k(a, c) + Q^{\Uparrow}_k(b, c) + 1$$

$$Q^{\Uparrow}_k(\lambda a, c) = Q^{\Uparrow}_{k+1}(a, c) + 1$$

Lemma 45 If $a, b \in \Lambda$, the shortest $\sigma_{\Uparrow}$-derivation of $a[\Uparrow^k(b \cdot id)]$ to its $\sigma_{\Uparrow}$-nf has length $Q^{\Uparrow}_k(a, b)$.

Proof By induction on a controlling all the possible $\sigma_{\Uparrow}$-derivations.

Definition 46 For $k \geq 0$ and $i \geq 2$, we define $M^u : \Lambda \to \mathbb{N}$ by induction as follows:

$M^u(\mathtt{n}) = 1 \qquad M^u(ab) = M^u(a) + M^u(b) + 1 \qquad M^u(\lambda a) = M^u(a) + 1$

Lemma 47 For $a \in \Lambda$, every u-derivation of $\varphi^i_k a$ to its u-normal form has length $M^u(a)$.

Proof By induction on a noting that derivations of $\varphi^i_k a$ begin with reductions at the root since $a \in \Lambda$.

Lemma 48 For every $a, b \in \Lambda$, $k \geq 0$ there exists a u-derivation of $a\sigma^{k+1}b$ to its u-nf whose length is less than or equal to $Q^{\Uparrow}_k(a, b)$.

Proof By induction on a. The interesting case is $a = \mathtt{k+1}$ and the result follows from Lemmas 43, 47 and the fact $M^u(b) \leq M^{\Uparrow}_{0i}(b)$, which is easily proved by induction on b.

Theorem 49 λu is more adequate than $\lambda\sigma_{\Uparrow}$.

Proof Show that for $a \in \Lambda$, and a $\lambda\sigma_{\Uparrow}$-derivation $a \to_{Beta} b \twoheadrightarrow^m_{\sigma_{\Uparrow}} \sigma_{\Uparrow}(b)$ there exists $n \leq m$ where $a \to_{\sigma-gen} c \twoheadrightarrow^n_u u(c)$ by induction on a. The interesting case is $a = (\lambda d)e \to_{Beta} d[e \cdot id] \twoheadrightarrow^m \sigma_{\Uparrow}(d[e \cdot id])$. By Lemmas 45 and 48, $m \geq Q^{\Uparrow}_0(d, e)$ and there exists a derivation $d\,\sigma^1 e \twoheadrightarrow^n_u u(d\,\sigma^1 e)$ where $n \leq Q^{\Uparrow}_0(d, e)$.

Now, to check the second condition in Definition 24, it is easy to compute to 6 the length of the shortest simulation in $\lambda\sigma_{\Uparrow}$ (there are only 2 such simulations) of the β-reduction $(\lambda\lambda 2)1 \to \lambda 2$, whereas the only simulation of this reduction in λu has length 4.

2.3 λu is more adequate than $\lambda\upsilon$

We use the functions defined in Sections 2.1 and 2.2 to show λu is more adequate than $\lambda\upsilon$.

Lemma 50 For every $a, b \in \Lambda$, $i \geq 0$ there exists a u-derivation of $a\sigma^{i+1}b$ to its u-nf whose length is less than or equal to $Q^{\upsilon}_i(a, b)$.

Proof By induction on a. The interesting case is $a = \mathtt{i} + \mathtt{1}$ and the result follows from Corollary 36, Lemma 47 and the fact $M^u(b) \leq i(1+M^{\upsilon}_0(b))$, proved by induction on b.

Theorem 51 λu is more adequate than $\lambda\upsilon$.

Proof We prove that for every $a \in \Lambda$ and every $\lambda\upsilon$-derivation $a \to_{Beta} b \twoheadrightarrow^m_{\upsilon} \upsilon(b)$ there exists $n \leq m$ such that $a \to_{\sigma-gen} c \twoheadrightarrow^n_u u(c)$ by induction on a. The proof is analogous to the proof of Theorem 49. For the second condition, use again the β-reduction $(\lambda\lambda 2)1 \to \lambda 2$ (see Theorem 49). It is easy to check that the only simulation of this in $\lambda\upsilon$ has length 5.

2.4 λu is more adequate than λs

The proof of adequacy in this section is simpler than the previous ones since λu and λs are closely related. We need first a lemma whose proof is by an easy induction on b:

Lemma 52 For $i \geq 2$ and $b \in \Lambda$ every s-derivation of $\varphi^i_0(b)$ to its s-nf is also a u-derivation.

Lemma 53 For every $a, b \in \Lambda$, $i \geq 1$ and s-derivation of $a\,\sigma^i b$ to its s-nf of length m, there exists an u-derivation of $a\,\sigma^i b$ to its u-nf whose length is less than or equal to m.

Proof By induction on a. The interesting case is $i > 1$ and $a = \mathtt{i}$. Note that the inequality is strict when $i = 1$ and $a = \mathtt{i}$. The result follows from Lemma 52 which gives a u-derivation of the same length.

Theorem 54 λu is more adequate than λs.

Proof Show, as in Theorem 49, that $\forall a \in \Lambda$ and $\forall \lambda s$-derivation $a \to_{\sigma-gen} b \twoheadrightarrow_s^m s(b)$, there exists $n \leq m$ where $a \to_{\sigma-gen} b \twoheadrightarrow_u^n u(c)$. To check the second condition, take $(\lambda 1)1 \to 1$. There is only one simulation in λs with length 4 and only one simulation in λu with length 3.

3 Non-comparable calculi

To show that two calculi, say $\lambda\xi_1$ and $\lambda\xi_2$ cannot be compared with our criterion it is enough to find two classical β-reductions $a \to_\beta b$ and $c \to_\beta d$ such that

1. There is a shorter simulation $a \twoheadrightarrow_{\lambda\xi_1} b$ than the shortest simulation $a \twoheadrightarrow_{\lambda\xi_2} b$.
2. There is a shorter simulation $c \twoheadrightarrow_{\lambda\xi_2} d$ than the shortest simulation $c \twoheadrightarrow_{\lambda\xi_1} d$.

If this is the case we say that $\lambda\xi_1$ and $\lambda\xi_2$ are *incomparable*, and we write $\lambda\xi_1 \not\prec\!\!\succ \lambda\xi_2$.

Since $\lambda\sigma$ works in a more "atomized" way (the $\Uparrow$-operator of $\lambda\sigma_\Uparrow$ and $\lambda\upsilon$ may be decomposed in $\lambda\sigma$ as $\Uparrow(s) = 1 \cdot (s \circ \uparrow)$ and the $/$-operator of $\lambda\upsilon$ may be decomposed in $\lambda\sigma$ as $a/ = a \cdot id$) it is tempting to assume that $\lambda\sigma$, even its version with uncoded de Bruijn indices, would be less adequate than $\lambda\upsilon$ and $\lambda\sigma_\Uparrow$. However this is not the case. As a matter of fact there is an infinite family of terms for which $\lambda\sigma$ performs better than $\lambda\upsilon$ and $\lambda\sigma_\Uparrow$, and furthermore, for these terms, $\lambda\sigma$ also performs better than λs and λu.

The terms we are going to consider are $(\lambda\lambda(2\,2))1^n$, where a^n is defined by induction on n as $a^1 = a$, $a^{n+1} = a\,a^n$. There is only one β-redex at the root and $(\lambda\lambda(2\,2))1^n \to_\beta \lambda(2^n 2^n)$. We study now the simulation of this β-reduction in the different calculi.

Lemma 55 There is a $\lambda\sigma$-derivation of $(\lambda\lambda(2\,2))1^n$ to its $\lambda\sigma$-nf whose length is $n+9$ and a $\lambda\sigma_{DB}$-derivation whose length is $2n+7$.

Proof Here is the derivation in $\lambda\sigma$:

$$(\lambda\lambda(2\,2))1^n = (\lambda\lambda(1[\uparrow]\,1[\uparrow]))1^n \to (\lambda(1[\uparrow]\,1[\uparrow]))[1^n \cdot id] \to \lambda((1[\uparrow]\,1[\uparrow])[1 \cdot ((1^n \cdot id)\circ \uparrow)]) \to$$
$$\lambda((1[\uparrow]\,1[\uparrow])[1 \cdot (1^n[\uparrow] \cdot (id\circ \uparrow))]) \twoheadrightarrow^{n-1} \lambda((1[\uparrow]\,1[\uparrow])[1 \cdot ((1[\uparrow])^n \cdot (id\circ \uparrow))]) \to$$
$$\lambda((1[\uparrow][1 \cdot (1[\uparrow])^n \cdot (id\circ\uparrow)])\,(1[\uparrow][1 \cdot (1[\uparrow])^n \cdot (id\circ\uparrow)])) \to$$
$$\lambda((1[\uparrow \circ (1 \cdot (1[\uparrow])^n \cdot (id\circ\uparrow))])\,(1[\uparrow][1 \cdot (1[\uparrow])^n \cdot (id\circ\uparrow)])) \to$$
$$\lambda((1[(1[\uparrow])^n \cdot (id\circ \uparrow)])\,(1[\uparrow][1 \cdot ((1[\uparrow])^n \cdot (id\circ \uparrow))])) \to \lambda((1[\uparrow])^n\,(1[\uparrow][1 \cdot ((1[\uparrow])^n \cdot (id\circ \uparrow))])) \twoheadrightarrow^3$$
$$\lambda((1[\uparrow])^n (1[\uparrow])^n) = \lambda(2^n 2^n)$$

Here is the derivation in $\lambda\sigma_{DB}$:

$$(\lambda\lambda(2\,2))1^n \to (\lambda(2\,2))[1^n \cdot id] \to \lambda((2\,2)[1 \cdot ((1^n \cdot id)\circ \uparrow)]) \to \lambda((2\,2)[1 \cdot (1^n[\uparrow] \cdot (id\circ \uparrow))]) \twoheadrightarrow^{n-1}$$
$$\lambda((2\,2)[1 \cdot ((1[\uparrow])^n \cdot (id\circ \uparrow))]) \twoheadrightarrow^n \lambda((2\,2)[1 \cdot (2^n \cdot (id\circ \uparrow))]) \to$$
$$\lambda((2[1 \cdot (2^n \cdot (id\circ \uparrow))])\,(2[1 \cdot (2^n \cdot (id\circ \uparrow))])) \to$$
$$\lambda((1[2^n \cdot (id\circ \uparrow)])\,(2[1 \cdot (2^n \cdot (id\circ \uparrow))])) \to \lambda(2^n\,(2[1 \cdot (2^n \cdot (id\circ \uparrow))])) \twoheadrightarrow^2 \lambda(2^n 2^n)$$

Lemma 56 Every $\lambda\upsilon$-derivation of $(\lambda\lambda(2\,2))1^n$ to its $\lambda\upsilon$-nf has length $4n+5$.

Proof Every derivation of $(\lambda\lambda(2\,2))1^n$ must begin as follows:

$$(\lambda\lambda(2\,2))1^n \to (\lambda(2\,2))[1^n/] \to \lambda((2\,2)[\Uparrow(1^n/)]) \to \lambda((2[\Uparrow(1^n/)])\,(2[\Uparrow(1^n/)]))$$

The two occurrences of $2[\Uparrow(1^n/)]$ cannot interact since no abstraction will appear in the first occurrence. Hence it is enough to show that every derivation of $2[\Uparrow(1^n/)]$ has length $2n+1$. This follows from $M_0^v(1^n) = 2n-1$ (easily shown by induction on n) and Lemma 39.

Lemma 57 Every λu-derivation of $(\lambda\lambda(2\,2))1^n$ to its λu-nf has length $4n+3$.

Proof Every λu-derivation of $(\lambda\lambda(2\,2))1^n$ must begin as follows:

$$(\lambda\lambda(2\,2))1^n \to (\lambda(2\,2))\sigma^1 1^n \to \lambda((2\,2)\sigma^2 1^n) \to \lambda((2\sigma^2 1^n)\,(2\sigma^2 1^n))$$

The two occurrences of $2\sigma^2 1^n$ cannot interact and hence it is enough to show that all derivations of $2\sigma^2 1^n$ have length $2n$. There is only one redex in $2\sigma^2 1^n$, whose contraction gives $\varphi_0^2(1^n)$ and by Lemma 47 every derivation of $\varphi_0^2(1^n)$ has length $M^u(1^n)$ which is easily computable to $2n-1$ by induction on n.

Lemma 58 For $a \in \Lambda$, every s-derivation of $\varphi_k^i a$ to its s-normal form has length $M^u(a)$.

Proof By induction on a. Identical to the proof of Lemma 47.

Lemma 59 Every λs-derivation of $(\lambda\lambda(2\,2))1^n$ to its λs-nf has length $4n+3$.

Proof Analogous to the proof of Lemma 57, using Lemma 58.

Lemma 60 There is a λt-derivation of $(\lambda\lambda(2\,2))1^n$ to its λt-nf whose length is $2n+4$.

Proof Here is the derivation in λt:

$$(\lambda\lambda(2\,2))1^n \to (\lambda(2\,2))\sigma^1 1^n \to \lambda((2\,2)\varsigma^2\theta_0(1^n)) \twoheadrightarrow^{n-1}$$
$$\lambda((2\,2)\varsigma^2(\theta_0 1)^n) \twoheadrightarrow^{n} \lambda((2\,2)\varsigma^2 2^n) \to \lambda((2\varsigma^2 2^n)\,(2\varsigma^2 2^n)) \twoheadrightarrow^{2} \lambda(2^n 2^n)$$

Lemma 61 The shortest $\lambda\sigma_{\Uparrow}$-derivation of $(\lambda\lambda(2\,2))1^n$ to its $\lambda\sigma_{\Uparrow}$-nf has length $4n+7$.

Proof Every $\lambda\sigma_{\Uparrow}$-derivation of $(\lambda\lambda(2\,2))1^n$ must begin as follows:

$$(\lambda\lambda(2\,2))1^n \to (\lambda(2\,2))[1^n\cdot id] \to \lambda((2\,2)[\Uparrow(1^n\cdot id)]) \to \lambda((2[\Uparrow(1^n\cdot id)])\,(2[\Uparrow(1^n\cdot id)]))$$

Now, the two occurrences of $2[\Uparrow(1^n\cdot id)]$ cannot interact and therefore, it is enough to verify that the shortest derivation of $2[\Uparrow(1^n\cdot id)]$ to its $\lambda\sigma_{\Uparrow}$-nf has length $2n+2$. This is easily done using Lemma 45 and the fact that $M_{01}^{\Uparrow}(1^n) = 2n-1$, proved by induction on n.

3.1 λu and λt are incomparable

Lemmas 57 and 60 prove that the reductions $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ with $n \geq 1$ show $\lambda u \not\preceq \lambda t$.

On the other hand, $(\lambda\lambda\lambda 3)1 \to \lambda\lambda 3$ shows that $\lambda t \not\preceq \lambda u$. In fact, it is easy to check that every simulation (there are 5) in λt of $(\lambda\lambda\lambda 3)1 \to \lambda\lambda 3$ has length 6, whereas in λu the unique simulation of this β-reduction has length 5.

3.2 λu and $\lambda\sigma$ are incomparable

Lemmas 57 and 55 prove that the reductions $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ with $n \geq 3$ show $\lambda u \not\preceq \lambda\sigma$ and $\lambda u \not\preceq \lambda\sigma_{DB}$. On the other hand, it is easy to show that $(\lambda 2)1 \to 1$ has unique simulations in λu, $\lambda\sigma$ and $\lambda\sigma_{DB}$ with respective lengths 2, 4 and 3. Hence, $\lambda\sigma \not\preceq \lambda u$ and $\lambda\sigma_{DB} \not\preceq \lambda u$.

3.3 λt and λs are incomparable

Lemmas 59 and 60 prove that the reductions $(\lambda\lambda(\mathtt{2}\,\mathtt{2}))\mathtt{1}^n \to \lambda(\mathtt{2}^n\mathtt{2}^n)$ with $n \geq 1$ show $\lambda s \not\preceq \lambda t$.

On the other hand, $(\lambda\lambda\lambda\mathtt{3})\mathtt{1} \to \lambda\lambda\mathtt{3}$ shows that $\lambda t \not\preceq \lambda s$. In fact, as in Section 3.1 it is easy to check that every simulation of this β-reduction in λs has length 5.

3.4 λt and $\lambda\sigma$ are incomparable

The simulation in λt of $(\lambda\mathtt{2})\mathtt{1} \to \mathtt{1}$ requires only 2 steps and hence (see Section 3.2) $\lambda\sigma \not\preceq \lambda t$ and $\lambda\sigma_{DB} \not\preceq \lambda t$. To show $\lambda t \not\preceq \lambda\sigma_{DB}$, take the β-reduction at the root of $(\lambda\lambda\lambda\lambda\mathtt{4})((\lambda\mathtt{1})(\lambda\mathtt{1}))$. It is possible to achieve the simulation in 19 steps in $\lambda\sigma_{DB}$ (let $s = ((\lambda\mathtt{1})(\lambda\mathtt{1}))\cdot id$):

$(\lambda\lambda\lambda\lambda\mathtt{4})((\lambda\mathtt{1})(\lambda\mathtt{1})) \to (\lambda\lambda\lambda\mathtt{4})[s] \twoheadrightarrow^3 \lambda\lambda\lambda(\mathtt{4}[\mathtt{1}\cdot((\mathtt{1}\cdot((\mathtt{1}\cdot(s\circ\uparrow))\circ\uparrow))\circ\uparrow)]) \to$
$\lambda\lambda\lambda(\mathtt{3}[(\mathtt{1}\cdot((\mathtt{1}\cdot(s\circ\uparrow))\circ\uparrow))\circ\uparrow]) \to \lambda\lambda\lambda(\mathtt{3}[\mathtt{1}[\uparrow]\cdot(((\mathtt{1}\cdot(s\circ\uparrow))\circ\uparrow)\circ\uparrow)]) \to$
$\lambda\lambda\lambda(\mathtt{2}[((\mathtt{1}\cdot(s\circ\uparrow))\circ\uparrow)\circ\uparrow]) \twoheadrightarrow^2 \lambda\lambda\lambda(\mathtt{2}[\mathtt{1}[\uparrow][\uparrow]\cdot(((s\circ\uparrow)\circ\uparrow)\circ\uparrow)]) \to$
$\lambda\lambda\lambda(\mathtt{1}[((s\circ\uparrow)\circ\uparrow)\circ\uparrow]) \twoheadrightarrow^2 \lambda\lambda\lambda(\mathtt{1}[s\circ\uparrow^3]) \to \lambda\lambda\lambda(\mathtt{1}[((\lambda\mathtt{1})(\lambda\mathtt{1}))[\uparrow^3]\cdot(id\circ\uparrow^3)]) \to$
$\lambda\lambda\lambda(((\lambda\mathtt{1})(\lambda\mathtt{1}))[\uparrow^3]) \to \lambda\lambda\lambda(((\lambda\mathtt{1})[\uparrow^3])((\lambda\mathtt{1})[\uparrow^3])) \twoheadrightarrow^2$
$\lambda\lambda\lambda((\lambda(\mathtt{1}[\mathtt{1}\cdot(\uparrow^3\circ\uparrow)]))\,(\lambda(\mathtt{1}[\mathtt{1}\cdot(\uparrow^3\circ\uparrow)]))) \twoheadrightarrow^2 \lambda\lambda\lambda((\lambda\mathtt{1})(\lambda\mathtt{1}))$

We must prove now that no simulation in λt of this β-reduction can be achieved in less than 19 steps. To do this we are going to prove a general result about λt. In Section 2.1 we have begun to study λt in order to compare it with $\lambda\upsilon$. Remark the analogy between Lemma 29 and Lemma 35 we aim now to a lemma which should correspond to Lemma 39, i.e. a result which will enable us to calculate the length of the t-derivations of $a\,\varsigma^i b$. Unfortunately, not all the derivations have the same length as for $\lambda\upsilon$. Furthermore, there is no easy way to compute the length of the shortest derivation as for $\lambda\sigma_{\Uparrow}$ (see Lemma 45). Hence, it does not seem easy to obtain such a general result. However, the shortest derivation of $a\,\varsigma^i b$ can always be calculated when a does not contain applications (like our example) and we proceed now to show it. The notions used here were introduced in Section 2.1.

Definition 62 We define $N : \Lambda_\theta \to \mathbb{N}$ recursively as follows:
$N(\mathtt{n}) = 0 \qquad N(ab) = N(a) + N(b) \qquad N(\lambda a) = N(a) \qquad N(\theta_k a) = M^t(a)$

Lemma 63 For $a \in \Lambda_\theta$, every t-derivation of a to its t-nf has length $N(a)$.

Proof By induction on the weight $P(b)$ used to prove SN for the t-calculus and case analysis. The proof is analogous to the proof of Lemma 29.

Definition 64 Let $\Lambda^- ::= \mathbb{N} \mid \lambda\Lambda^-$, i.e. Λ^- is the set of λ-terms which do not contain applications. For $i \geq 1$, we define $Q^t_i : \Lambda^- \times \Lambda_\theta \to \mathbb{N}$ by induction as follows:

$$Q^t_i(\mathtt{n}, b) = \begin{cases} 1 & \text{if } n \neq i \\ N(b) + 1 & \text{if } n = i \end{cases}$$

$$Q^t_i(\lambda a, b) = Q^t_{i+1}(a, \theta_0 b) + 1$$

Lemma 65 For $a \in \Lambda^-$, $b \in \Lambda_\theta$ and $i \geq 1$ the shortest derivation of $a\,\varsigma^i b$ to its t-nf has length $Q^t_i(a, b)$.

Proof Analogous to the proof of Lemma 29 using Lemma 63 for the case $a = \mathtt{i}$.

Now, since our simulation starts as $(\lambda\lambda\lambda\lambda 4)((\lambda 1)(\lambda 1)) \to (\lambda\lambda\lambda 4)\varsigma^1((\lambda 1)(\lambda 1))$, we use the previous lemma to conclude that every simulation of the β-reduction at the root has length 20. Therefore, $\lambda t \not\preccurlyeq \lambda\sigma_{DB}$.

3.5 λt and $\lambda\sigma_{\Uparrow}$ are incomparable

The simulation in $\lambda\sigma_{\Uparrow}$ of $(\lambda 2)1 \to 1$ requires 4 steps and hence (see Section 3.4) $\lambda\sigma_{\Uparrow} \not\preccurlyeq \lambda t$.

To show $\lambda t \not\preccurlyeq \lambda\sigma_{\Uparrow}$ we use the results of the previous subsection and the fact that there is a simulation in $\lambda\sigma_{\Uparrow}$ of the β-reduction at the root in $(\lambda\lambda\lambda\lambda 4)((\lambda 1)(\lambda 1))$ whose length is 14. Here it is (we denote again $s = ((\lambda 1)(\lambda 1)) \cdot id$):

$(\lambda\lambda\lambda\lambda 4)((\lambda 1)(\lambda 1)) \to (\lambda\lambda\lambda 4)[s] \twoheadrightarrow^3 \lambda\lambda\lambda(4[\Uparrow^3(s)] \twoheadrightarrow^3 \lambda\lambda\lambda(1[s \circ \uparrow^3]) \to$
$\lambda\lambda\lambda(1[((\lambda 1)(\lambda 1))[\uparrow^3] \cdot (id \circ \uparrow^3)]) \to \lambda\lambda\lambda(((\lambda 1)(\lambda 1))[\uparrow^3]) \to$
$\lambda\lambda\lambda(((\lambda 1)[\uparrow^3])((\lambda 1)[\uparrow^3])) \twoheadrightarrow^2 \lambda\lambda\lambda((\lambda(1[\Uparrow(\uparrow^3)]))(\lambda(1[\Uparrow(\uparrow^3)]))) \twoheadrightarrow^2 \lambda\lambda\lambda((\lambda 1)(\lambda 1))$

3.6 λs and $\lambda\sigma$ are incomparable

Lemmas 59 and 55 prove that the reductions $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ with $n \geq 3$ show $\lambda s \not\preccurlyeq \lambda\sigma$ and $\lambda s \not\preccurlyeq \lambda\sigma_{DB}$. On the other hand, it is immediate to verify that $(\lambda 2)1 \to 1$ has a unique simulation in λs of length 2 and hence (see Section 3.2) $\lambda\sigma \not\preccurlyeq \lambda s$ and $\lambda\sigma_{DB} \not\preccurlyeq \lambda s$.

3.7 λs and $\lambda\sigma_{\Uparrow}$ are incomparable

It is immediate to verify that $(\lambda 1)1 \to 1$ has unique simulations in λs and $\lambda\sigma_{\Uparrow}$ of respective lengths 3 and 2. Therefore, $\lambda s \not\preccurlyeq \lambda\sigma_{\Uparrow}$. On the other hand, the simulations in λs and $\lambda\sigma_{\Uparrow}$ of $(\lambda 2)1 \to 1$ (see Sections 3.5 and 3.6) show that $\lambda\sigma_{\Uparrow} \not\preccurlyeq \lambda s$.

3.8 λs and $\lambda\upsilon$ are incomparable

The reduction $(\lambda\lambda 2)1 \to \lambda 2$ has unique simulations in λs and $\lambda\upsilon$ of respective lengths 4 and 5. Therefore, $\lambda\upsilon \not\preccurlyeq \lambda s$. On the other hand, $(\lambda 1)1 \to 1$ has a unique simulation in $\lambda\upsilon$ of length 2 and hence (see Section 3.7) $\lambda s \not\preccurlyeq \lambda\upsilon$.

3.9 $\lambda\sigma$ and $\lambda\upsilon$ are incomparable

Lemmas 56 and 55 prove that the reductions $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ with $n \geq 2$ show $\lambda\upsilon \not\preccurlyeq \lambda\sigma$ and $\lambda\upsilon \not\preccurlyeq \lambda\sigma_{DB}$. On the other hand, it is easy to verify that the shortest simulation in $\lambda\sigma$ (there are only 9), resp. $\lambda\sigma_{DB}$ (there are only 5), of $(\lambda\lambda 2)1 \to \lambda 2$ has length 7, resp. 6, and hence (see Section 3.8) $\lambda\sigma \not\preccurlyeq \lambda\upsilon$ and $\lambda\sigma_{DB} \not\preccurlyeq \lambda\upsilon$.

3.10 $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$ are incomparable

Lemmas 61 and 55 prove that the reductions $(\lambda\lambda(2\,2))1^n \to \lambda(2^n 2^n)$ with $n \geq 1$ show $\lambda\sigma_{\Uparrow} \not\preccurlyeq \lambda\sigma$ and $\lambda\sigma_{\Uparrow} \not\preccurlyeq \lambda\sigma_{DB}$. On the other hand, there is a simulation in $\lambda\sigma_{\Uparrow}$ of $(\lambda\lambda 3)1 \to \lambda 2$ of length 7:

$(\lambda\lambda 3)1 \to (\lambda 3)[1 \cdot id] \to \lambda(3[\Uparrow(1 \cdot id)]) \to \lambda(2[(1 \cdot id)\circ\uparrow]) \to$
$\lambda(2[1[\uparrow] \cdot (id\circ\uparrow)]) \to \lambda(1[id\circ\uparrow]) \to \lambda(1[\uparrow]) \to \lambda 2$

whereas it is easy to check that every simulation (there are only 14) in $\lambda\sigma$ of this β-reduction has length 8. Therefore, $\lambda\sigma \not\prec \lambda\sigma_{\Uparrow}$.

Unfortunately, the previous example does not work to show $\lambda\sigma_{DB} \not\prec \lambda\sigma_{\Uparrow}$. It is easy to find a simulation in $\lambda\sigma_{\Uparrow}$ of $(\lambda\lambda\lambda 3)1 \to \lambda\lambda 3$ of length 9. However, in $\lambda\sigma_{DB}$ every simulation of this β-reduction has length at least 11. This can be checked by hand or a simple program can do the work.

3.11 $\lambda\sigma_{\Uparrow}$ and $\lambda\upsilon$ are incomparable

The shortest simulation (there are only 2) in $\lambda\sigma_{\Uparrow}$ of $(\lambda\lambda 2)1 \to \lambda 2$ has length 6 and hence (see Section 3.8) $\lambda\sigma_{\Uparrow} \not\prec \lambda\upsilon$. On the other hand, there is a $\lambda\sigma_{\Uparrow}$-simulation of $(\lambda\lambda\lambda\lambda 4)(1\,1) \to \lambda\lambda\lambda(4\,4)$ of length 16:

$(\lambda\lambda\lambda\lambda 4)(1\,1) \to (\lambda\lambda\lambda 4)[(1\,1)\cdot id] \twoheadrightarrow^3 \lambda\lambda\lambda(4[\Uparrow^3((1\,1)\cdot id)]) \twoheadrightarrow^3 \lambda\lambda\lambda(1[((1\,1)\cdot id)\circ\uparrow^3]) \to$
$\lambda\lambda\lambda(1[(1\,1)[\uparrow^3]\cdot(id\circ\uparrow^3)]) \to \lambda\lambda\lambda((1\,1)[\uparrow^3]) \to \lambda\lambda\lambda(1[\uparrow^3]\,1[\uparrow^3]) \twoheadrightarrow^6 \lambda\lambda\lambda(4\,4)$

whereas the length of every simulation in $\lambda\upsilon$ can be easily evaluated to 17: in fact, every derivation must start as: $(\lambda\lambda\lambda\lambda 4)(1\,1) \to (\lambda\lambda\lambda 4)[(1\,1)/]$ and then apply Lemma 39 with $i = 0$. Therefore, $\lambda\upsilon \not\prec \lambda\sigma_{\Uparrow}$.

We summarize in the following table the results obtained so far. The table must be entered from the left, thus the information given, for instance, in position $(1,3)$ is to be read as $\lambda u \prec \lambda s$, whereas the information in position $(3,1)$ is $\lambda s \succ \lambda u$.

	λu	λt	λs	$\lambda\upsilon$	$\lambda\sigma$	$\lambda\sigma_{\Uparrow}$
λu	$=$	$\not\prec\not\succ$	$\prec$	$\prec$	$\not\prec\not\succ$	$\prec$
λt	$\not\prec\not\succ$	$=$	$\not\prec\not\succ$	$\prec$	$\not\prec\not\succ$	$\not\prec\not\succ$
λs	$\succ$	$\not\prec\not\succ$	$=$	$\not\prec\not\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$
$\lambda\upsilon$	$\succ$	$\succ$	$\not\prec\not\succ$	$=$	$\not\prec\not\succ$	$\not\prec\not\succ$
$\lambda\sigma$	$\not\prec\not\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$	$=$	$\not\prec\not\succ$
$\lambda\sigma_{\Uparrow}$	$\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$	$\not\prec\not\succ$	$=$

4 The bridging calculi

4.1 The $\lambda\omega$-calculus

In order to express λs-terms in the $\lambda\sigma$-style we are going to split the closure operator of $\lambda\sigma$ (denoted in a semi-infix notation as $-[-]$) in a family of closures operators that shall be denoted also with a semi-infix notation as $-[-]_i$, where i ranges on the set of natural numbers.

We will admit as basic operators the iterations of $\uparrow$ and therefore we will have a countable set of basic substitutions $\uparrow^n$, where n ranges on the set of natural numbers. By doing so, the updating operators of λs are available in our new syntax as $-[\uparrow^n]_i$.

σ-generation	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b/]_1$
σ-app-transition	$(a\, b)[s]_j$	$\longrightarrow$	$(a\,[s]_j)\,(b\,[s]_j)$
σ-λ-transition	$(\lambda a)[s]_j$	$\longrightarrow$	$\lambda(a[s]_{j+1})$
σ-/-destruction	$\mathtt{n}[a/]_j$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if } n > j \\ a[\uparrow^{j-1}]_1 & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases}$
σ-↑-destruction	$\mathtt{n}[\uparrow^i]_j$	$\longrightarrow$	$\begin{cases} \mathtt{n+i} & \text{if } n \geq j \\ \mathtt{n} & \text{if } n < j \end{cases}$

Figure 9: The $\lambda\omega$-calculus

Finally, we introduce a *slash* operator of sort `term` $\rightarrow$ `substitution` which transform a term a into a substitution $a/$. This operator may be considered as *consing with id* (in the $\lambda\sigma$-jargon) and has been exploited in the $\lambda\upsilon$-calculus (cf. [BBLRD96]).

Here is the formalisation of this syntax and the rewriting rules of $\lambda\omega$:

Definition 66 The set of terms of the $\lambda\omega$-calculus, noted $\Lambda\omega$, is defined as $\Lambda\omega^t \cup \Lambda\omega^s$, where $\Lambda\omega^t$ and $\Lambda\omega^s$ are defined by the following mutual recursion:

$$\begin{array}{ll} \textbf{Terms} & \Lambda\omega^t ::= \mathbb{N} \mid \Lambda\omega^t \Lambda\omega^t \mid \lambda\Lambda\omega^t \mid \Lambda\omega^t[\Lambda\omega^s]_j \\ \textbf{Substitutions} & \Lambda\omega^s ::= \uparrow^i \mid \Lambda\omega^t/ \end{array}$$

where $j \geq 1$ and $i \geq 0$. The set, denoted $\lambda\omega$, of rules of the *λω-calculus* is given in Figure 9.

The set of rules of the *ω-calculus* is $\lambda\omega - \{\sigma{-}generation\}$. We use $a, b, c, \ldots$ to range over $\Lambda\omega^t$ and $s, t, \ldots$ to range over $\Lambda\omega^s$.

As we said before, the /-operator is present in $\lambda\upsilon$. Furthermore, the constant substitutions $\uparrow^i$ are exploited in the calculus of Muñoz [Muñ97b] $\lambda\phi$. This calculus is so designed to avoid the non left linear rule *(SCons)* of $\lambda\sigma_{SP}$. Moreover, our indexed substitutions are reminiscent of the substitutions of the $\lambda\chi$-calculus with *levels* considered in [LRD95].

However, there is an essential difference between $\lambda\omega$ and $\lambda\chi$: in $\lambda\chi$ the terms (which are described with variable names) are stratified in levels whereas this is not the case for the $\lambda\omega$-terms. There is also an essential difference between $\lambda\phi$ and $\lambda\omega$ concerning the substitutions: composition is a basic operator in $\lambda\phi$ but it does not exist in $\lambda\omega$.

It is interesting to realize that the iterations $\uparrow^i$ as basic operators as well as the indexed substitutions are features which are embodied in λs since, as we shall prove in the next section, $\lambda\omega$ and λs are isomorphic.

σ-/-transition	$a\,[b/]_k[s]_j$	$\longrightarrow$	$a\,[s]_{j+1}[b[s]_{j-k+1}/]_k$	if $k \le j$
/-$\uparrow$-transition	$a\,[\uparrow^i]_k[b/]_j$	$\longrightarrow$	$\begin{cases} a[b/]_{j-i}[\uparrow^i]_k & \text{if } k+i \le j \\ a[\uparrow^{i-1}]_k & \text{if } k \le j < k+i \end{cases}$	
$\uparrow$-$\uparrow$-transition	$a\,[\uparrow^i]_k[\uparrow^l]_j$	$\longrightarrow$	$\begin{cases} a[\uparrow^l]_{j-i}[\uparrow^i]_k & \text{if } k+i < j \\ a[\uparrow^{i+l}]_k & \text{if } k \le j \le k+i \end{cases}$	

Figure 10: The new rules of the $\lambda\omega_e$-calculus

4.2 The $\lambda\omega_e$-calculus

As we pointed out in Section 1.3 the λs-calculus is not even locally confluent on open terms. The same negative result can be easily transferred to the $\lambda\omega$-calculus.

By open terms in this new syntax we mean terms which admit variables (usually called metavariables) of sort `term` but not metavariables of sort `substitution`. In the $\lambda\sigma$-jargon they are often referred as *semi-closed* or pure terms (cf. [Río93]).

Now, we define formally what we mean by open terms in our new syntax and give the $\lambda\omega_e$-rules:

Definition 67 The set of *open terms*, noted $\Lambda\omega_{op}$ is defined as $\Lambda\omega^t_{op} \cup \Lambda\omega^s_{op}$, where $\Lambda\omega^t_{op}$ and $\Lambda\omega^s_{op}$ are defined by the following mutual recursion:

$$\begin{array}{ll} \textbf{Open Terms} & \Lambda\omega^t_{op} ::= \mathbf{V} \mid \mathbb{N} \mid \Lambda\omega^t_{op}\Lambda\omega^t_{op} \mid \lambda\Lambda\omega^t_{op} \mid \Lambda\omega^t_{op}[\Lambda\omega^s_{op}]_j \\ \textbf{Substitutions} & \Lambda\omega^s_{op} ::= \uparrow^i \mid \Lambda\omega^t_{op}/ \end{array}$$

where $j \ge 1$ and $i \ge 0$, and where $\mathbf{V}$ stands for a set of variables, over which $X, Y, \ldots$ range. We take a, b, c to range over $\Lambda\omega^t_{op}$ and s, t, $\ldots$ over $\Lambda\omega^s_{op}$. The set, denoted $\lambda\omega_e$, of rules of the $\lambda\omega_e$-*calculus* is obtained by adding to the set of rules $\lambda\omega$ the new rules of Figure 10.

The set of rules of the ω_e-*calculus* is $\lambda\omega_e - \{\sigma\textit{-generation}\}$.

Remark that the rule schemes /-$\uparrow$ and $\uparrow$-$\uparrow$ can be merged into the single scheme

$$a\,[\uparrow^i]_k[s]_j \to a[s]_{j-i}[\uparrow^i]_k \qquad \text{for } k+i < j$$

but they must be kept distinct for the case $k+i=j$ if SN is to be preserved. In fact, the $\uparrow$-$\uparrow$-scheme, if admitted in the case $k+i=j$, may generate an infinite loop by itself (take for instance $i=k=l=1$ and $j=2$).

5 The isomorphisms

We define in this section two functions, that are inverse of each other, and that establish an isomorphism between λs_e and $\lambda\omega_e$. Furthermore, their restriction to ground terms also establishes an isomorphism between λs and $\lambda\omega$. These isomorphisms translate all the properties of λs and λs_e to $\lambda\omega$ and $\lambda\omega_e$, respectively. We remark that the sets of terms Λs and Λs_{op} correspond with the sets of terms $\Lambda\omega^t$ and $\Lambda\omega^t_{op}$, respectively, rather than $\Lambda\omega$ and $\Lambda\omega_{op}$. Thus, it is only the sort `term` that is involved in the isomorphism.

Definition 68 The functions $T : \Lambda s_{op} \to \Lambda\omega^t_{op}$ and $S : \Lambda\omega^t_{op} \to \Lambda s_{op}$ are defined inductively by:

$$\begin{array}{ll}
T(X) = X & S(X) = X \\
T(\mathtt{n}) = \mathtt{n} & S(\mathtt{n}) = \mathtt{n} \\
T(a\,b) = T(a)T(b) & S(a\,b) = S(a)S(b) \\
T(\lambda a) = \lambda T(a) & S(\lambda a) = \lambda S(a) \\
T(a\,\sigma^j b) = T(a)[T(b)/]_j & S(a\,[b/]_j) = S(a)\,\sigma^j S(b) \\
T(\varphi^i_k a) = T(a)[\uparrow^{i-1}]_{k+1} & S(a\,[\uparrow^i]_k) = \varphi^{i+1}_{k-1}(S(a))
\end{array}$$

We make an "abus de notation" and use the same names T and S for the restrictions of these functions to ground terms. The context will be always clear enough in order to avoid ambiguities.

Lemma 69 The following hold:

1. For all $a, b \in \Lambda s$, if $a \to_s b$ then $T(a) \to_\omega T(b)$.
2. For all $a, b \in \Lambda s$, if $a \to_{\lambda s} b$ then $T(a) \to_{\lambda\omega} T(b)$.
3. For all $a, b \in \Lambda s_{op}$, if $a \to_{s_e} b$ then $T(a) \to_{\omega_e} T(b)$.
4. For all $a, b \in \Lambda s_{op}$, if $a \to_{\lambda s_e} b$ then $T(a) \to_{\lambda\omega_e} T(b)$.

Proof By induction on a: if the reduction is internal, the IH applies; otherwise, the theorem must be manually checked for each rule.

Lemma 70 The following hold:

1. For all $a, b \in \Lambda\omega^t$, if $a \to_\omega b$ then $S(a) \to_s S(b)$.
2. For all $a, b \in \Lambda\omega^t$, if $a \to_{\lambda\omega} b$ then $S(a) \to_{\lambda s} S(b)$.
3. For all $a, b \in \Lambda\omega^t_{op}$, if $a \to_{\omega_e} b$ then $S(a) \to_{s_e} S(b)$.
4. For all $a, b \in \Lambda\omega^t_{op}$, if $a \to_{\lambda\omega_e} b$ then $S(a) \to_{\lambda s_e} S(b)$.

Proof By induction on a: if the reduction is internal, the IH applies; otherwise, the theorem must be manually checked for each rule.

We verify finally that T and S are in fact inverse of each other.

Lemma 71 The following hold:

1. For all $a \in \Lambda\omega^t$, we have $T(S(a)) = a$.
2. For all $a \in \Lambda\omega^t_{op}$, we have $T(S(a)) = a$.

3. For all $a \in \Lambda s$, we have $S(T(a)) = a$.
4. For all $a \in \Lambda s_{op}$, we have $S(T(a)) = a$.

Proof By an easy induction on a.

Now that the calculi have been proved isomorphic, all the results of sections 1.3 and 1.4 concerning λs and λs_e translate into corresponding results for the sort `term` to $\lambda\omega$ and $\lambda\omega_e$.

Lemma 72 The following hold:

1. The ω-calculus is SN and confluent on $\Lambda\omega^t$.
2. Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda\omega} b$ then $a \twoheadrightarrow_{\beta} b$.
3. Let $a, b \in \Lambda$, if $a \rightarrow_{\beta} b$ then $a \twoheadrightarrow_{\lambda\omega} b$.
4. The $\lambda\omega$-calculus is confluent on $\Lambda\omega^t$.
5. Pure terms which are SN in the λ-calculus are also SN in the $\lambda\omega$-calculus.

Proof Use the isomorphism and the corresponding results for λs summarized in Theorem 12.

Lemma 73 The following hold:

1. The ω_e-calculus is weakly normalising and confluent.
2. The $\lambda\omega_e$-calculus is confluent on open terms.
3. Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda\omega_e} b$ then $a \twoheadrightarrow_{\beta} b$.
4. Let $a, b \in \Lambda$, if $a \rightarrow_{\beta} b$ then $a \twoheadrightarrow_{\lambda\omega_e} b$.

Proof Use the isomorphism, Lemma 19 and Theorems 18, 20 and 21.

Remark that the schemes *σ-σ-tr.* and *φ-σ-tr.* of λs_e both translate in the same scheme of $\lambda\omega_e$, namely *σ-/-transition*.

6 Typed calculi

We begin with a brief survey of the typed versions of λs and $\lambda\sigma$. From the point of view of syntax the only difference is that the abstractions are marked with types. We have thus $\lambda A.a$ where A is a simple type, that is a type obtained from a set of basic types using the only binary infix constructor of types $\rightarrow$. In the case of $\lambda\sigma$ we also have the conses marked with types: $a : A \cdot s$.

We recall that environments in de Bruijn's setting are simply lists of types and in the case of $\lambda\sigma$, substitutions receive environments as types. We introduce the following notation concerning environments. If E is the environment $E_1, E_2, \ldots, E_n$, we shall use the notation $E_{\geq i}$ for the environment $E_i, E_{i+1}, \ldots, E_n$, analogously $E_{\leq i}$ stands for $E_1, \ldots, E_i$, etc.

The rewriting rules of the corresponding typed calculi are exactly the same (except that rules involving abstractions are now typed).

6.1 The typing rules

We concentrate now on the typing rules of these calculi. We begin by recalling the typing rules for the simply typed λ-calculus in de Bruijn's notation. We call the typing system **L1**:

$$(\mathbf{L1} - \mathit{var}) \qquad A, E \vdash \mathtt{1} : A \qquad\qquad (\mathbf{L1} - \lambda) \qquad \frac{A, E \vdash b : B}{E \vdash \lambda A.b : A \to B}$$

$$(\mathbf{L1} - \mathit{varn}) \qquad \frac{E \vdash \mathtt{n} : B}{A, E \vdash \mathtt{n+1} : B} \qquad\qquad (\mathbf{L1} - \mathit{app}) \qquad \frac{E \vdash b : A \to B \quad E \vdash a : A}{E \vdash b\, a : B}$$

We recall now the typing rules for λs and λs_e. The typing system **Ls1** is defined as follows:

The rules **Ls1**-*var*, **Ls1**-*varn*, **Ls1**-λ and **Ls1**-*app* are exactly the same as **L1**-*var*, **L1**-*varn*, **L1**-λ and **L1**-*app*, respectively. The new rules are:

$$(\mathbf{Ls1} - \sigma) \qquad \frac{E_{\geq i} \vdash b : B \quad E_{<i}, B, E_{\geq i} \vdash a : A}{E \vdash a\, \sigma^i b : A} \qquad\qquad (\mathbf{Ls1} - \varphi) \qquad \frac{E_{\leq k}, E_{\geq k+i} \vdash a : A}{E \vdash \varphi^i_k a : A}$$

In order that the reader could compare with the typing system $\mathbf{L}\sigma\mathbf{1}$ of $\lambda\sigma$, we recall $\mathbf{L}\sigma\mathbf{1}$:

The rules $\mathbf{L}\sigma\mathbf{1}$-*var*, $\mathbf{L}\sigma\mathbf{1}$-$\lambda$ and $\mathbf{L}\sigma\mathbf{1}$-*app* are exactly the same as **L1**-*var*, **L1**-λ and **L1**-*app*, respectively. The new rules are:

$$(\mathbf{L}\sigma\mathbf{1} - \mathit{clos}) \qquad \frac{E \vdash s \triangleright E' \quad E' \vdash a : A}{E \vdash a[s] : A} \qquad\qquad (\mathbf{L}\sigma\mathbf{1} - \mathit{id}) \qquad E \vdash \mathit{id} \triangleright E$$

$$(\mathbf{L}\sigma\mathbf{1} - \mathit{cons}) \qquad \frac{E \vdash a : A \quad E \vdash s \triangleright E'}{E \vdash a : A \cdot s \triangleright A, E'} \qquad\qquad (\mathbf{L}\sigma\mathbf{1} - \mathit{shift}) \qquad A, E \vdash \uparrow \triangleright E$$

$$(\mathbf{L}\sigma\mathbf{1} - \mathit{comp}) \qquad \frac{E \vdash s'' \triangleright E'' \quad E'' \vdash s' \triangleright E'}{E \vdash s' \circ s'' \triangleright E'} \qquad\qquad (\mathbf{L}\sigma\mathbf{1} - \mathit{Mtv}) \qquad E_X \vdash X : A_X$$

The last rule is added to type open terms and should be understood as follows: for every metavariable X, there exists an environment E_X and a type A_X such that the rule holds.

We introduce now the typing rules for $\lambda\omega$ and $\lambda\omega_e$. The typing system is called $\mathbf{L}\omega\mathbf{1}$. The rules $\mathbf{L}\omega\mathbf{1}$-*var*, $\mathbf{L}\omega\mathbf{1}$-*varn*, $\mathbf{L}\omega\mathbf{1}$-$\lambda$ and $\mathbf{L}\omega\mathbf{1}$-*app* are exactly the same as **L1**-*var*, **L1**-*varn*, **L1**-λ and **L1**-*app*, respectively. The new rules are:

$$(\mathbf{L}\omega\mathbf{1} - \mathit{id}) \qquad E \vdash \uparrow^0 \triangleright E \qquad\qquad (\mathbf{L}\omega\mathbf{1} - \mathit{slash}) \qquad \frac{E \vdash a : A}{E \vdash a/ : A, E}$$

$$(\mathbf{L}\omega\mathbf{1} - \mathit{shift}) \qquad \frac{E \vdash \uparrow^i \triangleright E'}{A, E \vdash \uparrow^{i+1} : E'} \qquad\qquad (\mathbf{L}\omega\mathbf{1} - \mathit{Mtv}) \qquad E_X \vdash X : A_X$$

$$(\mathbf{L}\omega\mathbf{1} - \mathit{clos}) \qquad \frac{E_{\geq j} \vdash s \triangleright E' \qquad E_{<j}, E'_1, E_{\geq j} \vdash a : A}{E \vdash a[s]_j : A}$$

We prove now that the isomorphism defined in Section 3 preserves typing. For the definition of T and S in the next theorem we refer to Section 3.

Theorem 74 The following hold:

1. For $a \in \Lambda s$, if $E \vdash a : A$ then $E \vdash T(a) : A$.
2. For $a \in \Lambda s_{op}$, if $E \vdash a : A$ then $E \vdash T(a) : A$.
3. For $a \in \Lambda\omega^t$, if $E \vdash a : A$ then $E \vdash S(a) : A$.
4. For $a \in \Lambda\omega^t_{op}$, if $E \vdash a : A$ then $E \vdash S(a) : A$.

Proof The four items are proved by an easy induction on the inference of $E \vdash a : A$.

6.2 Subject Reduction

This section is devoted to establish Subject Reduction for our four calculi. We prove first subject reduction for $\lambda\omega$ and $\lambda\omega_e$ and then we use the isomorphisms given in the previous section to obtain Subject Reduction for λs and λs_e.

Theorem 75 (Subject Reduction for $\lambda\omega$) Let $a, b \in \Lambda\omega^t$ and $s, t \in \Lambda\omega^s$.
1. If $E \vdash a : A$ and $a \to_{\lambda\omega} b$ then $E \vdash b : A$.
2. If $E \vdash s \triangleright F$ and $s \to_{\lambda\omega} t$ then $E \vdash t \triangleright F$.

Proof By simultaneous induction on the structure of a and s. If the reduction is internal it is enough to apply the inductive hypothesis. If the reduction is at the root then each rule must be examined. We check for instance the rule *σ-/-destruction* for the case $n = j$.

Let us assume $E \vdash \mathtt{n}[a/]_j : A$. Therefore there exists an environment E' such that $E_{\geq j} \vdash a/ \triangleright E'$ and $E_{<j}, E'_1, E_{\geq j} \vdash \mathtt{n} : A$. Hence the n-th type in the environment $E_{<j}, E'_1, E_{\geq j}$ is A.

From $E_{\geq j} \vdash a/ \triangleright E'$ we deduce $E_{\geq j} \vdash a : E'_1$ and, since $A = (E_{<j}, E'_1, E_{\geq j})_n$ and $n = j$, we have $A = E'_1$. Therefore, $E_{\geq j} \vdash a : A$ and, because $E \vdash \uparrow^{j-1} \triangleright E_{\geq j}$, we can apply the *clos*-rule (remember $E = E_{\geq 1}$ and, by convention, $E_{<1} = nil$) to obtain $E \vdash a[\uparrow^{j-1}]_1 : A$.

Theorem 76 (Subject Reduction for $\lambda\omega_e$) Let $a, b \in \Lambda\omega^t_{op}$ and $s, t \in \Lambda\omega^s_{op}$.
1. If $E \vdash a : A$ and $a \to_{\lambda\omega_e} b$ then $E \vdash b : A$.
2. If $E \vdash s \triangleright F$ and $s \to_{\lambda\omega_e} t$ then $E \vdash t \triangleright F$.

Proof By simultaneous induction on the structure of a and s. The proof is analogous to the previous proof, only the new rules must be checked now. As an example we study the rule *$\sigma - /$-transition*.

Assume $E \vdash a[b/]_k[s]_j : A$ and $k \leq j$. Therefore, there exists an environment E' such that

$$E_{\geq j} \vdash s \triangleright E' \tag{1}$$

and $E_{<j}, E'_1, E_{\geq j} \vdash a[b/]_k : A$. From this last equation we deduce the existence of an environment E'' such that

$$E_{<k}, E''_1, E_k, \ldots, E_{j-1}, E'_1, E_{\geq j} \vdash a : A \tag{2}$$

and $E_k, \ldots, E_{j-1}, E'_1, E_{\geq j} \vdash b / \rhd E''$. Therefore,

$$E_k, \ldots, E_{j-1}, E'_1, E_{\geq j} \vdash b : E''_1 \tag{3}$$

Applying the *clos* rule, from equations 1 and 2 we get

$$E_{<k}, E''_1, E_{\geq k} \vdash a[s]_{j+1} : A \tag{4}$$

and from equations 1 and 3, $E_{\geq k} \vdash b[s]_{j-k+1} : E''_1$, and a further application of *slash* gives

$$E_{\geq k} \vdash b[s]_{j-k+1} / : E''_1, E_{\geq k} \tag{5}$$

Finally, applying *clos* to equations 4 and 5, we conclude

$$E \vdash a[s]_{j+1}[b[s]_{j-k+1}/]_k : A$$

We use now the translations to prove Subject Reduction for λs and λs_e.

Theorem 77 (Subject Reduction for λs and λs_e) Let $a, b \in \Lambda s$ and $c, d \in \Lambda s_{op}$.

1. If $E \vdash a : A$ and $a \to_{\lambda s} b$ then $E \vdash b : A$.
2. If $E \vdash c : A$ and $c \to_{\lambda s_e} d$ then $E \vdash d : A$.

Proof We just check the first item (the second is analogous).

If $E \vdash a : A$ then, by Lemma 74.1, $E \vdash T(a) : A$. On the other hand, if $a \to_{\lambda s} b$ then, by Lemma 69.2, $T(a) \to_{\lambda\omega} T(b)$. Now, by Theprem 75.1, $E \vdash T(b) : A$, and by Lemma 74.2, we get $E \vdash S(T(b)) : A$, and we are done because $S(T(b)) = b$, by Lemma 71.3.

Finally, we mention that in [KRW98], we showed that every well typed term in the λs-calculus is strongly normalising. This implies due to the above isomorphism that every well typed term in the $\lambda\omega$-calculus is strongly normalising. Also, in [KR99] we show that every well typed term in the simply typed $\lambda\omega_e$-calculus is weakly normalising. This again implies that every well typed term in the simply typed λs_e-calculus is weakly normalising.

Conclusions

In this paper, we attempted to bridge and compare the two styles of explicit substitutions: those à la $\lambda\sigma$ and those à la λs. We did this in two steps:

1. We introduced a criterion of adequacy to simulate β-reduction in calculi of explicit substitutions and we applied it to several calculi: $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\upsilon$, λs, λt and λu. The latter is presented here for the first time and may be considered as an adequate variant of λs. By doing so, we established that calculi à la λs are usually more adequate at simulating β-reduction than calculi in the $\lambda\sigma$-style. We showed that λt is more adequate than $\lambda\upsilon$ and that λu is more adequate than $\lambda\upsilon$, $\lambda\sigma_{\Uparrow}$ and λs and gave counterexamples to show that all other comparisons are impossible. We are aware that our criterion is a very basic one, and it would be interesting to study the relation among the different calculi in terms of complexity of the length of reductions (linear, exponential). However, we consider our results as a first step in the study of adequacy.

2. We introduced two new calculi $\lambda\omega$ and $\lambda\omega_e$ that can bridge the two styles of calculi of explicit substitutions. Our motivation for doing so comes from the fact that the two different styles of substitutions provide complementary properties and so it is interesting to understand one style in terms of the other. Another reason is that, the λs-style still has one puzzling open problem: the termination of the substitution calculus s_e. Although, the $\lambda\omega$ and $\lambda\omega_e$-calculi are calculi in the $\lambda\sigma$-style, their stratified substitutions are more λs-style than $\lambda\sigma$-style. The main new feature towards the $\lambda\sigma$-style is the introduction of $\uparrow^i$ and hence the availability of the updated term $\varphi^i_k(a)$ as $a[\uparrow^{i-1}]_{k+1}$. Hence the stratified substitutions play a double role: as *real substitutions* as in $a[b/]_k$ and as *updatings* as in $a[\uparrow^i]_k$. We believe that this two-sorted presentation of λs may be useful to gain a new insight on the open problem for this calculus, mainly the strong normalisation of s_e.

   Apart from their role as bridging calculi between the $\lambda\sigma$- and λs-styles of explicit substitutions, the $\lambda\omega$- and $\lambda\omega_e$-calculi are interesting on their own for the following reasons:

   (a) $\lambda\omega$ is confluent on closed terms and preserves strong normalisation,

   (b) the associated calculus of substitutions of $\lambda\omega$ is SN,

   (c) the simply typed version of $\lambda\omega$ is SN,

   (d) $\lambda\omega$ possesses an extension $\lambda\omega_e$ that is confluent on open terms, simulates β-reduction, and whose simply typed version is weakly normalising (on open term).

   As far as we know, the $\lambda\omega$-calculus is the first calculus in the $\lambda\sigma$-style that has all those properties. However, the preservation of strong normalisation does not hold for $\lambda\omega_e$ and the SN of the associated calculus of substitution of $\lambda\omega_e$ remains unsolved.

Acknowledgements

The authors are grateful for useful feedback and discussions with Roel Bloo, Jan Willem Klop and Pierre Lescanne. This work is supported by EPSRC grants number GR/L15685 and GR/L36963.

References

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

[AS86] H. Abelson and G. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1986.

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

[BBLRD96] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a Calculus of Explicit Substitutions which Preserves Strong Normalization. *Journal of Functional Programming*, 6(5):699–722, 1996.

[Ben97] Z. Benaissa. *Les calculs de substitutions explicites comme fondement de l'implantation des langages fonctionnels.* PhD thesis, Univ. Henri Poincare, Nancy, 1997.

[Blo97] R. Bloo. *Preservation of Strong Normalisation for Explicit Substitution.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1997.

[Blo99] R. Bloo. Pure type systems with explicit substitutions. *In proceedings of FLOC'99 workshop WESTAPP'99*, pages 45–58, 1999.

[Bon99] E. Bonelli. The polymorphic lambda calculus with explicit substitutions. *In proceedings of FLOC'99 workshop WESTAPP'99*, pages 59–74, 1999.

[BR96] R. Bloo and K. Rose. Combinatory Reduction Systems with Explicit Substitution that Preserve Strong Normalisation. *Proceedings of RTA '96, Lecture Notes in Computer Science*, 1103, 1996.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[Con86] R. Constable et al. *Implementing Mathematics with the NUPRL Development System.* Prentice-Hall, 1986.

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

[dB72] 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.

[dB78] 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.

[DHK95] G. Dowek, T. Hardin, and C. Kirchner. Higher order unification via explicit substitutions (extended abstract). In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science,* San Diego, pages 366–374, 1995.

[FKP99] Maria C. F. Ferreira, Delia Kesner, and Laurence Puel. λ-calculi with explicit substitutions preserving strong normalization. *Applicable Algebra in Engineering, Communication and Computation*, 9(4):333–371, 1999.

[GL97] Jean Goubault-Larrecq. A proof of weak termination of the simply typed $\lambda\sigma$-calculus. Technical report, INRIA, January 1997.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[Gui99a] B. Guillaume. The λ_l-calculus. *In proceedings of FLOC'99 workshop WESTAPP'99*, pages 2–13, 1999.

[Gui99b] B. Guillaume. *Un calcul des substitutions avec etiquettes*. PhD thesis, Université de Savoie, Chambéry, 1999.

[HL89] T. Hardin and J.-J. Lévy. A Confluent Calculus of Substitutions. *France-Japan Artificial Intelligence and Computer Science Symposium*, December 1989.

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

[KR95] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proceedings of Programming Languages Implementation and the Logic of Programs PLILP'95*, volume 982 of *Lecture Notes in Computer Science*, pages 45–62. Springer-Verlag, 1995.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with Explicit Substitution which preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7(4):395–420, 1997.

[KR98] F. Kamareddine and A. Ríos. Bridging de Bruijn indices and variable names in explicit substitutions calculi. *The Logic Journal of the Interest Group of Pure and Applied Logic, IGPL*, 6(6):843–874, 1998.

[KR99] F. Kamareddine and A. Ríos. Weak normalisation of the simply typed λs_e-calculus. Technical report, Heriot-Watt University, 1999. In preparation.

[KRW98] F. Kamareddine, A. Ríos, and J.B. Wells. Calculi of generalised β_e-reduction and explicit substitution: Type free and simply typed versions. *Journal of Functional and Logic Programming*, pages 1 – 44, 1998.

[Les94] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In Hans Boehm, editor, *Proceedings of the 21st Annual ACM Symposium on Principles Of Programming Languages, Portland (Or., USA)*, pages 60–69. ACM, 1994.

[LM96] J. L. Lawall and H. Mairson. Optimality and inefficiency: What isn't a cost model of the lambda calculus? Proc. 1996 ACM SIGPLAN Int'l Conf. Functional Programming, pages 92–101, 1996.

[LR98] F. Lang and K. Rose. Two calculi of explicit substitution with confluence on metaterms and preservation of strong normalisation. *In proceedings of RTA workshop WESTAPP'98*, 1998.

[LRD95] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn's levels. In J. Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, volume 914 of *Lecture Notes in Computer Science*, pages 294–308. Springer-Verlag, 1995.

[Mag95] Magnusson. *The implementation of ALF - a proof editor based on Martin Löf's Type Theory with explicit substitutions*. PhD thesis, Chalmers, 1995.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *Proceedings of Typed Lambda Calculi and Application: TLCA'95*, volume 902 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Muñ96] C. Muñoz. Proof representation in type theory: State of the art. In *Proceedings of the XXII Latin-American Conference of Informatics CLEI Panel 96*, Santafé de Bogotá, Colombia, June 1996.

[Muñ97a] C. Muñoz. A calculus of substitutions for incomplete-proof representation in type theory. Technical Report RR-3309, Unité de recherche INRIA-Rocquencourt, Novembre 1997.

[Muñ97b] C. Muñoz. Dependent types with explicit substitutions: A meta-theoretical development. In *Types for Proofs and Programs, Proceedings of the International Workshop TYPES'96*, volume 1512 of *Lecture Notes in Computer Science*, pages 294–316, 1997.

[Muñ97c] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types*. Thèse de doctorat, Université Paris 7, 1997. English version is available as an INRIA research report number RR-3309.

[Muñ98] C. Muñoz. Proof synthesis via explicit substitutions on open terms. In *Proc. International Workshop on Explicit Substitutions, Theory and Applications, WESTAPP 98*, Tsukuba (Japan), April 1998.

[NGdV94] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*. North-Holland, Amsterdam, 1994.

[NW90] G. Nadathur and D. Wilson. A representation of lambda terms suitable for operations on their intentions. *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 341–348, 1990.

[Pau90] L. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[Pey87] S.L. Peyton-Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, 1987.

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