Explicit Substitutions with de Bruijn's levels

Pierre LESCANNE and Jocelyne ROUYER-DEGLI

Centre de Recherche en Informatique de Nancy (CNRS) and INRIA-Lorraine
Campus Scientifique, BP 239,
F54506 Vandœuvre-lès-Nancy, France

email: {Pierre.Lescanne, Jocelyne.Rouyer}@loria.fr

1 Introduction

In the introduction of [7], Curry writes that substitution is the main issue in logic and that λ-calculus does not properly answer the problem because substitutions are outside the calculus. He pleads in favor of combinatory logic which offers a full treatment to substitutions through its use of a first order rewrite system in which substitution is cleanly handled. However, one may object that this formal system is not as natural as λ-calculus for describing the concept of function. In 1972, de Bruijn [8] proposed two notations he calls *indices* and *levels* that avoid α-conversion, i.e., renaming in terms, and later in 1978 [9, 10], he described a calculus based on his indices which nowadays we would call *explicit substitutions* and which proposes a full and correct treatment of substitution. Since that time, most of the formalisms for describing λ-calculus and explicit substitutions are based on de Bruijn's indices. Unlike our predecessors, in this paper we want to use de Bruijn's levels.

Calculus of explicit substitutions is a λ-calculus in which substitution is not external but is fully integrated at the same level as β reduction. This internalisation of the substitution calculus is achieved by rewrite rules which allow a full and easy mechanism for describing β reduction. The original goal of explicit substitutions is to provide the implementor of AUTOMATH (and later of functional programming languages) with a finer granularity in the description of the process of substitutions, as substitutions play the central role in the implementation of those systems and languages. This way, controls that postpone costly operations may be adopted and operations that will turn out to be unnecessary will never be performed *(lazy evaluation)*.

Except for a sketched attempt in [1] (*$\lambda\sigma$-calculus with names*), all the proposed calculi use *De Bruijn indices*. That notation has two drawbacks. First, terms are hard to read due to the use of numbers instead of names and due to the fact that the "same" variable is designated by different numbers according to the context. Second the association between variables and their values, the so-called environment, keeps changing whenever one leaves or enters an abstraction. To our knowledge "levels" were only mentioned twice in the literature, first by de Bruijn [8] and later by Crégut [4], but never in the framework of explicit substitutions. Both authors suggest a canonical indexing of the variables;

Crégut calls it "reversed De Bruijn indexing". To make formulas as readable as in the classical λ-calculus, we give each variable a name made from its index and that name is the same everywhere in a pure term, i.e., a term without *closure* (see below) unlike classical De Bruijn index. Two important features of $\lambda\chi$ are the absence of variables of type *substitution* and the absence of composition of substitutions. Of course, the λ-calculus describes variables, but they are seen as constants by the rewrite system $\lambda\chi$ and they will be called *names* in what follows. The only variables of $\lambda\chi$ (those which play an actual role in the rewrite system) are of type *Term* and of type *Nat*.

To illustrate our approach and to allow the reader to make comparisons, Figure 1 gives a few examples in different notations. The first line is in the usual notation, the second is in $\lambda\chi$ notation, the third is in notation with de Bruijn's indices and the fourth is in level indices.

Yg	
Classical:	$(\lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx)))\ g$
Lambda Chi:	$(\lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)))\ x_{-1}$
De Bruijn indices:	$(\lambda(\lambda(\underline{1}(\underline{0}\,\underline{0}))\ \lambda(\underline{1}(\underline{0}\,\underline{0}))))\ g$
level indices:	$(\lambda(\lambda\underline{0}(\underline{1}\,\underline{1}))\,(\lambda\underline{0}(\underline{1}\,\underline{1})))\ g$

Succ	
Classical:	$\lambda x \cdot x(\lambda y \cdot x\,y)$
Lambda Chi:	$\lambda x_0 \cdot x_0(\lambda x_1 \cdot x_0\,x_1)$
De Bruijn indices:	$\lambda\underline{0}(\lambda\underline{1}\,\underline{0})$
level indices:	$\lambda\underline{0}(\lambda\underline{0}\,\underline{1})$

Fig. 1. A few examples making notations explicit

Restrictions are imposed on names. In each term, each λ receives a level. λ's at the highest level (level 0) are associated with x_0 and λ's at level i are associated with x_i. Free variables, those that are not bound to any λ receive negative subscripts. In this framework, we can describe substitution and β reduction.

2 χ-terms, rules and examples

Let us call χ-terms, terms with explicit canonical variables and DB-terms, terms with De Bruijn's indices. The key to the correctness of $\lambda\chi$ is a translation from the first to the second. This translation relies on two operators σ_j and τ_i^j introduced for describing β reduction in DB-terms [4]. First let us introduce the important concept of level in χ-terms. In what follows Term_i ($i \geq 0$) is the set of subterms of χ-terms at level i. A term at level i lies under i symbols λ and can contain only variables with indices up to $i-1$. The set of all χ-terms is Term_0.

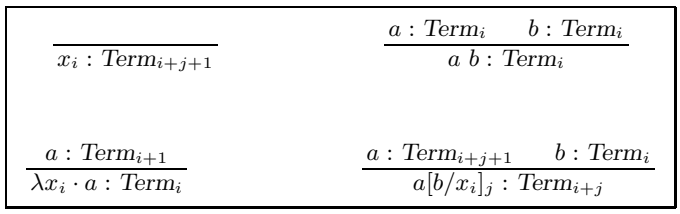


Fig. 2. The level system: a description of χ-terms

The grammar of χ-terms is described by a set of inference rules, which we call the *level system*.

Before giving the rules, let us look at the reduction of terms in this calculus.

$$\begin{aligned}
\mathsf{Y} \quad &\equiv \quad \lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)) && (1)\\
&\xrightarrow[B]{} \quad \lambda x_0 \cdot (x_0(x_1x_1))[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0 && (2)\\
&\xrightarrow[\chi]{+} \quad \lambda x_0 \cdot x_0[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0\,(x_1[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0\,x_1[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0) && (3)\\
&\xrightarrow[\chi]{} \quad \lambda x_0 \cdot x_0\ (x_1[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0\ x_1[\lambda x_1 \cdot x_0(x_1x_1)/x_1]_0) && (4)\\
&\xrightarrow[\chi]{+} \quad \lambda x_0 \cdot x_0\ (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)) && (5)
\end{aligned}$$

Rewrite (2) creates an explicit substitution which when distributed through the whole term $x_0(x_1x_1)$ implements the β reduction. Rewrites (3) distribute the substitution through the term. Rewrite (4) performs the substitution on subterm x_0, i.e., leaves x_0 unchanged. Rewrites (5) perform the substitutions on subterm x_1.

$$\begin{aligned}
\mathsf{Y}\ \mathsf{Y} \quad &\equiv \quad (\lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)))\ \mathsf{Y} && (6)\\
&\xrightarrow[B]{} \quad ((\lambda x_1 \cdot x_0(x_1x_1))\ (\lambda x_1 \cdot x_0(x_1x_1)))[\mathsf{Y}/x_0]_0 && (7)\\
&\xrightarrow[\chi]{} \quad (\lambda x_1 \cdot x_0(x_1x_1))[\mathsf{Y}/x_0]_0\ (\lambda x_1 \cdot x_0(x_1x_1))[\mathsf{Y}/x_0]_0 && (8)\\
&\xrightarrow[\chi]{} \quad (\lambda x_0 \cdot (x_0(x_1x_1))[\mathsf{Y}/x_0]_1)\ (\lambda x_1 \cdot x_0(x_1x_1))[\mathsf{Y}/x_0]_0 && (9)\\
&\xrightarrow[\chi]{+} \quad (\lambda x_0 \cdot (x_0[\mathsf{Y}/x_0]_1\ (x_1[\mathsf{Y}/x_0]_1\ x_1[\mathsf{Y}/x_0]_1)))\ (\lambda x_1 \cdot x_0(x_1x_1))[\mathsf{Y}/x_0]_0 && (10)\\
&\xrightarrow[\chi]{+} \quad (\lambda x_0 \cdot (x_0[\mathsf{Y}/x_0]_1\ (x_0\ x_0)))\ (\lambda x_1 \cdot x_0(x_1x_1))[\mathsf{Y}/x_0]_0 && (11)
\end{aligned}$$

Rewrite (9) pushes the substitution under λ and so increments by 1 the index of the substitution and changes the names associated with this λ. Rewrite (10) distributes that substitution and rewrite (11) performs the substitutions on the x_1's which renames them to x_0.

In what a follows terms of the form $a[b/x_i]_j$ are called *closures,* terms of the form $a\ b$ are called *applications,* terms of the form $\lambda x_i \cdot a$ are called *abstractions.*

From the above examples, we notice a few facts. Each substitution carries an index which we call the *depth* of the substitution. When a substitution $[a/x_i]_j$ goes through a λ, its index grows and the name associated with that λ decreases. When one actually performs a substitution, i.e., when one reduces a term of the form $x_i[a/x_k]_j$, if $i > k$ the index i of x_i is decremented, if $i < k$ the name is left unchanged. An actual replacement takes place only when a reduction of a term of the form $x_i[a/x_i]_j$ is performed. For instance, let us consider a reduction of the term $\lambda x_0 \cdot (x_0[\mathsf{Y}/x_0]_1\ (x_0\ x_0)))$. Performing the substitution $[\mathsf{Y}/x_0]_1$ requires replacement of the names x_0 by Y, in which names are modified. Its λx_0 will become a λx_1 and its λx_1 will become a λx_2 and its bound variables have to be modified accordingly. Therefore

$$\begin{aligned}
&\lambda x_0 \cdot (x_0[\mathsf{Y}/x_0]_1\ (x_0\ x_0)) \\
&\quad \equiv \lambda x_0 \cdot (x_0[\lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1))/x_0]_1)\ (x_0\ x_0) \\
&\quad \xrightarrow[\chi]{} \lambda x_0 \cdot (rename(\lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)), 0, 1)\ (x_0\ x_0)) \\
&\quad \xrightarrow[\chi]{+} \lambda x_0 \cdot (\lambda x_1 \cdot (\lambda x_2 \cdot x_1(x_2x_2))\,(\lambda x_2 \cdot x_1(x_2x_2)))(x_0\,x_0).
\end{aligned}$$

A function *rename* is introduced and in $rename(a, i, j)$, a is the term to be renamed, i is the level of the variable that created the renaming and j is the distance between a and the level where the substitution was created. In the previous case, we have

$$\begin{aligned}
&rename(\lambda x_0 \cdot (\lambda x_1 \cdot x_0(x_1x_1))\,(\lambda x_1 \cdot x_0(x_1x_1)), 0, 1) \\
&\qquad \xrightarrow[\chi]{+} \\
&\lambda x_1 \cdot (\lambda x_2 \cdot x_1(x_2x_2))\,(\lambda x_2 \cdot x_1(x_2x_2)).
\end{aligned}$$

Rename can be seen as an "explicit" α conversion. Its behaviour is easily described by rewrite rules as well as the rest of the calculus (Fig. 3). The following rule needs to be added to the level system

$$\boxed{\frac{a : Term_{i+k}}{rename(a, i, j) : Term_{i+k+j}}}$$

to deal with *rename*. The system without the rule B, i.e., the system $\lambda\chi \setminus \{B\}$ that deals only with substitution removal, is called χ. The *normal form* of a is written $\chi(a)$, but we also use the notation $a \xrightarrow[\chi]{!} b$ to say that b is the nomal form of a. $\lambda\chi$ preserves levels (a kind of subject reduction). In rule $\mathsf{Var}_>$, one has $0 \leq k < j$; that condition is naturally enforced by the level system. Moreover, $\lambda\chi$ uses the operator $+$ which is associative and commutative, such rewrite systems with associativity and commutativity are well known [2, 11] and allow us to avoid conditional rules. More precisely, we prefer to write

$$x_{i+k+1}[a/x_i]_j \ \rightarrow\ x_{i+k}$$

rather than

$$\frac{k > i}{x_k[a/x_i]_j \ \rightarrow\ x_{k-1}}$$

although in some proofs we prefer the equivalent second view. χ is not left-linear, but variables of sort $Term_i$ (denoted by a, b and c) occur always once in left-hand sides (we will take advantage of this property) and the system has no critical pairs. In the next section we show that it correctly implements β reduction. χ does not contain composition of substitutions. There is a debate on its introduction. For some people it is a feature to have and for other it is a feature not to have. For us, the most natural way to handle this operation is by "concatenation" of substitutions getting expressions like $a[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$. Anyway calculi of explicit substitutions which contain composition are not strongly normalising on typed terms and we feel this feature is important.

(B)	$(\lambda x_i \cdot a)b \to a[b/x_i]_0$
(App)	$(a\ b)[c/x_i]_j \to a[c/x_i]_j\ b[c/x_i]_j$
(Lambda)	$(\lambda x_{i+j+1} \cdot a)[b/x_i]_j \to \lambda x_{i+j} \cdot (a[b/x_i]_{j+1})$
($\mathsf{Var}_>$)	$x_{i+k+1}[a/x_i]_j \to x_{i+k}$
($\mathsf{Var}_<$)	$x_i[a/x_{i+k+1}]_j \to x_i$
($\mathsf{Var}_=$)	$x_i[a/x_i]_j \to rename(a,i,j)$
(RenApp)	$rename(a\ b,i,j) \to rename(a,i,j)\ rename(b,i,j)$
(RenLambda)	$rename(\lambda x_{i+k} \cdot a,i,j) \to \lambda x_{i+k+j} \cdot rename(a,i,j)$
($\mathsf{RenVar}_\geq$)	$rename(x_{i+k},i,j) \to x_{i+k+j}$
($\mathsf{RenVar}_<$)	$rename(x_i,i+k+1,j) \to x_i$

Fig. 3. The calculus of explicit substitutions with levels $\lambda\chi$

3 Termination of χ

Consider the morphism:

$$\begin{aligned} \kappa : \quad \lambda x_i \cdot a &\mapsto \Lambda(\kappa(a)) \\ a\ b &\mapsto App(\kappa(a),\kappa(b)) \\ a[b/x_i]_j &\mapsto C(\kappa(a),\kappa(b)) \\ rename(a,i,j) &\mapsto ren(\kappa(a)) \\ x_i &\mapsto x \end{aligned}$$

and the system χ_1 of Fig.4. If $a \xrightarrow[\chi]{} b$ then $\kappa(a) \xrightarrow[\chi_1]{} \kappa(b)$.

Termination (or strong normalisation) of χ_1 implies this of χ and is proved by the recursive path ordering $>_{rpo}$ derived from the precedence $C > ren > \Lambda > App > x$. Remember that x is considered as a "constant".

Let us define rewsub an ordering based on $\xrightarrow[\chi]{}$ and used in forthcoming proofs.

(App')	$C(App(a,b),c) \to App(C(a,c),C(b,c))$
(Lambda')	$C(\Lambda a,b) \to \Lambda(C(a,b))$
($\mathsf{Var}_>$')	$C(x,a) \to x$
($\mathsf{Var}_<$')	$C(x,a) \to x$
($\mathsf{Var}_=$')	$C(x,a) \to ren(a)$
(RenApp')	$ren(a\ b) \to ren(a)\ ren(b)$
(RenLambda')	$ren(\Lambda a) \to \Lambda(ren(a))$
($\mathsf{RenVar}_\geq$')	$ren(x) \to x$
($\mathsf{RenVar}_<$')	$ren(x) \to x$

Fig. 4. The rewrite system χ_1 for the proof of termination of χ

Definition 1 (The rewsub ordering). *The* rewsub *ordering is the ordering*

$$(\xrightarrow[\chi]{} \cup \sqsupset)^+,$$

where $\sqsupset$ is the subterm relation.

Rewsub is a well-founded ordering since $\xrightarrow[\chi]{}$ is simply terminating [15], i.e., its termination is proved by a simplification ordering. Rewsub is actually the smallest simplification ordering that contains $\xrightarrow[\chi]{}$.

4 Translation and correctness

As we have seen in the previous examples, β reduction is described by $a \xrightarrow[\beta]{} b$ *if and only if* $a \xrightarrow[B]{} b'$ *and* $b = \chi(b')$ *where is* $\chi(b')$ *is the normal form of* b' *by the rewrite system* χ. In this section, we prove the correctness of this definition. For this, we connect it with a definition known to be correct, namely a definition given in terms of De Bruijn indices [4, 5] and based on the two basic functions σ_n and τ_i^n.

$$\begin{array}{c}\sigma_n(ac,b) = \sigma_n(a,b)\sigma_n(c,b) \\ \sigma_n(\lambda a,b) = \lambda(\sigma_{n+1}(a,b))\end{array} \quad \sigma_n(\underline{m},b) = \begin{cases} \underline{m-1} \text{ if } m > n+1 \\ \tau_0^n(b) \text{ if } m = n+1 \\ \underline{m} \text{ if } m \leq n \end{cases}$$

where:

$$\begin{array}{c}\tau_i^n(ab) = \tau_i^n(a)\tau_i^n(b) \\ \tau_i^n(\lambda a) = \lambda(\tau_{i+1}^n(a))\end{array} \quad \tau_i^n(\underline{m}) = \begin{cases} \underline{m+n} \text{ if } m > i \\ \underline{m} \text{ if } m \leq i \end{cases}$$

$T(\_,i)$ is a translation of χ-terms of level i to DB-terms.

$$\begin{aligned} T(a\ b,i) &= T(a,i)\ T(b,i) \\ T(\lambda x_i \cdot a,i) &= \lambda T(a,i+1) \\ T(x_i,i+k+1) &= \underline{k+1} \\ T(a[b/x_i]_j,i+j) &= \sigma_j(T(a,i+j+1),T(b,i)) \\ T(rename(a,i,j),i+j+k) &= \tau_k^j(T(a,i+k)) \end{aligned}$$

The main proposition says that χ preserves T.

Proposition 1. *1. If $a \xrightarrow[\chi]{} b$ then if a exists at level i, then b exists at level i and $T(a,i) = T(b,i)$.*
2. $T((\lambda x_i \cdot a)b, i) \xrightarrow[\beta]{} T(a[b/x_i]_0, i) = T(\chi(a[b/x_i]_0), i)$.

Proof. We have to check the first assertion for each of the nine rules of χ at the correct level. The following equalities can be verified without difficulty using the definitions and the properties of T, σ_n and τ_i^n.

App

$$\begin{aligned} T((a\ b)[c/x_i]_j, i+j) &= \sigma_j(T(a, i+j+1), T(c,i))\ \sigma_j(T(b, i+j+1), T(c,i)) \\ &= T(a[c/x_i]_j b[c/x_i]_j, i+j) \end{aligned}$$

Lambda

$$\begin{aligned} T((\lambda x_{i+j+1} \cdot a)[c/x_i]_j, i+j) &= \lambda(\sigma_{j+1}(T(a, i+j+2), T(c,i))) \\ &= T(\lambda x_{i+j} \cdot (a[c/x_i]_{j+1}), i+j) \end{aligned}$$

$\mathsf{Var}_>$

$$\begin{aligned} T(x_{i+k+1}[a/x_i]_j, i+j) &= \underline{j-k} \\ &= T(x_{i+k}, i+j) \quad \text{for } k < j \end{aligned}$$

$\mathsf{Var}_<$

$$\begin{aligned} T(x_i[a/x_{i+k+1}]_j, i+k+1+j) &= \underline{k+j+1} \\ &= T(x_i, i+k+1+j) \end{aligned}$$

$\mathsf{Var}_=$

$$\begin{aligned} T(x_i[a/x_i]_j, i+j) &= \tau_0^j(T(a,i)) \\ &= T(rename(a,i,j), i+j) \end{aligned}$$

RenApp

$$\begin{aligned} T(rename(a\ b, i, j), i+j+k) &= \tau_k^j(T(a, i+k))\ \tau_k^j(T(b, i+k)) \\ &= T(rename(a,i,j)\ rename(b,i,j), i+j+k) \end{aligned}$$

RenLambda

$$\begin{aligned} T(rename(\lambda x_{i+k} \cdot, i, j), i+k+j) &= \lambda(\tau_{k+1}^j(T(a, i+k+1))) \\ &= T(\lambda x_{i+k+j} \cdot rename(a,i,j), i+k+j) \end{aligned}$$

$\mathsf{RenVar}_\leq$

$$\begin{aligned} T(rename(x_{i+k}, i, j), i+k+l+1+j) &= \underline{l+1} \\ &= T(x_{i+k+j}, i+k+j+l+1) \end{aligned}$$

$\mathsf{RenVar}_<$

$$T(rename(x_i, i+k+1, j), i+k+l+1+j) = \underline{k+1+l+j}$$
$$= \overline{T(x_i, i+k+1+l+j)}$$

For proving assertion 2, we use assertion 1.

$$\begin{array}{rcl} T((\lambda x_i \cdot a)\ b, i) & = & T(\lambda x_i \cdot a, i)\ T(b, i) \\ & = & \lambda(T(a, i+1)\ T(b, i)) \\ & \xrightarrow[\beta]{} & \sigma_0(T(a, i+1), T(b, i)) \\ & = & T(a[b/x_i]_0, i) \\ & & \qquad \text{by definition of } T \\ & = & T(\chi(a[b/x_i]_0, i)) \\ & & \qquad \text{by assertion 1.} \end{array}$$

5 Confluence of $\lambda\chi$

The goal of this section is to prove the confluence of $\lambda\chi$ over closed terms (terms without variables). The *confluence theorem* is based on a lemma classical when using Hardin's interpretation method which we call the *projection lemma*. It itself requires the $\lambda\chi$ version of an important classical lemma the *substitution lemma*. Before proving the substitution lemma, we state routine lemmas on *rename* expressions, namely Lemma 1 to Lemma 5, whose proofs by induction are straightforward providing Lemma 5 is invoked after Lemma 4.

The relation $\overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}}$ is χ-convertibility. Since χ is confluent, it is defined by $a \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} b$ if there exists c such that $a \xrightarrow[\chi]{*} c$ and $b \xrightarrow[\chi]{*} c$. In every proof of convertibility below, we will only examine terms where variables stand for terms in χ-normal form, i.e., terms without substitutions or *rename*. Indeed this will not change the generality of the result.

Lemma 1.

$$rename(rename(b, i+k, p), i, j) \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} rename(rename(b, i, j), i+k+j, p)$$

Lemma 2. $rename(a[b/x_{i+k}]_p, i, j) \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} rename(a, i, j)[rename(b, i, j)/x_{i+k+j}]_p$.

Lemma 3. $rename(c, p, i+j) \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} rename(c, p, i+j+1)[d/x_{i+p}]_{j+q}$.

Lemma 4. $rename(c, p, i+k+j) \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} rename(rename(c, p, i+k), p+i, j)$.

Lemma 5. $rename(b, i+p+1, j)[c/x_p]_{i+k+j} \overset{}{\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow}} rename(b[c/x_p]_{i+k}, i+p, j)$.

Lemma 6. *(Substitution lemma)*

$$a[b/x_{i+p+1}]_j[c/x_p]_{i+j} \quad \underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow} \quad a[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j.$$

Proof. This lemma is proved by structural induction on a. The level of a is $i+p+j+2$.

- $a \equiv \lambda x_{i+p+j+2} \cdot a'$.
  $(\lambda x_{i+p+j+2} \cdot a')[b/x_{i+p+1}]_j[c/x_p]_{i+j}$
  $$\begin{aligned}
  &\xrightarrow[\chi]{*} && \lambda x_{i+p+j} \cdot (a'[b/x_{i+p+1}]_{j+1}[c/x_p]_{i+j+1}) \\
  &\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow} && \lambda x_{i+p+j} \cdot (a'[c/x_p]_{i+j+2}[b[c/x_p]_i/x_{i+p}]_{j+1}) \\
  &\xrightarrow[\lambda\chi]{*} && (\lambda x_{i+p+j+2} \cdot a')[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j
  \end{aligned}$$
- $a \equiv a'\ b'$
  $(a'\ b')[b/x_{i+p+1}]_j[c/x_p]_{i+j}$
  $$\begin{aligned}
  &\xrightarrow[\chi]{} && a'[b/x_{i+p+1}]_j[c/x_p]_{i+j}\ b'[b/x_{i+p+1}]_j[c/x_p]_{i+j} \\
  &\underset{\chi}{\twoheadleftarrow\!\!\!\twoheadrightarrow} && a'[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j\ b'[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j \\
  &\xrightarrow[\lambda\chi]{} && (a'\ b')[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j
  \end{aligned}$$
- $a \equiv x_k$ with $k < i+p+2$.
  We have to rewrite $x_k[b/x_{i+p+1}]_j[c/x_p]_{j+1}$ and $x_k[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j$. Only rules Var are applied and three cases have to be considered. Both terms rewrite in two steps to the same terms. Those terms are:
  - x_k for $k < p$,
  - x_{k-1} for $p < k < i+p+1$,
  - x_{k-2} for $k > i+p+1$.

  The other cases need lemmas. For $k = p$, we have

  $$x_k[b/x_{i+p+1}]_j[c/x_p]_{j+1} \quad \xrightarrow[\chi]{*} \quad rename(c, p, i+j)$$

  and

  $$x_k[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j \quad \xrightarrow[\chi]{*} \quad rename(c, p, i+j+1).$$

  and the result comes from Lemma 3.
  The last case $k = i+p+1$ gives us

  $$x_k[b/x_{i+p+1}]_j[c/x_p]_{i+j} \quad \xrightarrow[\chi]{} \quad rename(b, i+p+1, j)[c/x_p]_{i+j}$$

  and

  $$x_k[c/x_p]_{i+j+1}[b[c/x_p]_i/x_{i+p}]_j \quad \xrightarrow[\chi]{*} \quad rename(b, [c/x_p]_i, i+p, j)$$

  the result is a consequence of Lemma 5.

Lemma 7 (Projection Lemma). *If* $a \xrightarrow[B]{} b$ *then* $\chi(a) \xrightarrow[\beta]{*} \chi(b)$.

Proof. The proof is very similar to that of [1] and [17]. The terms can be supposed of the form $a \equiv a'[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$ and $b \equiv b'[d_1/x_{i_1}]_{j_1} \dots [d_p/x_{i_p}]_{j_p}$ where a' is not a closure. We proceed by rewsub induction on a and we distinguish cases according to the structure of a'.

- $a \equiv (a_1 a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$ and $b \equiv (b_1 a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$ and the B redex occurs inside a_1 with $a_1 \xrightarrow[B]{} b_1$, then

$$a_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p} \xrightarrow[B]{} b_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}.$$

  By induction,

$$\chi(a_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \xrightarrow[\beta]{*} \chi(b_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p})$$

  and

$$\begin{array}{rcl}
\chi(a) & = & \chi((a_1 a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& = & \chi(a_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p})\chi(a_2[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& \xrightarrow[\beta]{*} & \chi(b_1[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p})\chi(a_2[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& = & \chi((b_1 a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& = & \chi(b).
\end{array}$$

  The case $a_2 \xrightarrow[B]{} b_2$ works similarly.
- If a' is the B redex, that is $a \equiv ((\lambda x_{i_0+p} \cdot a'_1)a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$ then b' is a closure, that is $b \equiv a'_1[a_2/x_{i_0+p}]_0[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$.

$$\begin{array}{rcl}
\chi(a) & = & \chi(((\lambda x_{i_0+p} \cdot a'_1)a_2)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& = & \lambda x_{i_0} \cdot (\chi(a'_1[c_1/x_{i_1}]_{j_1+1} \dots [c_p/x_{i_p}]_{j_p+1}))\ \chi(a_2[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) \\
& \xrightarrow[\beta]{*} & \chi(\chi(a'_1[c_1/x_{i_1}]_{j_1+1} \dots [c_p/x_{i_p}]_{j_p+1})[\chi(a_2[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p})/x_{i_0}]_0) \\
& = & \chi(a'_1[c_1/x_{i_1}]_{j_1+1} \dots [c_p/x_{i_p}]_{j_p+1}[a_2[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}/x_{i_0}]_0)
\end{array}$$

  and by repeating p times (zero times if $p = 0$) the Substitution Lemma

$$= \chi(a'_1[a_2/x_{i_0+p}]_0[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}) = \chi(b)$$

- If $a \equiv (\lambda x_{i_0} \cdot a_1)[c_1/x_{i_1}]_{j_1} \dots [c_p/x_{i_p}]_{j_p}$, then $a_1 \xrightarrow[B]{} b_1$ or $c_i \xrightarrow[B]{} d_i$,

$$a_1[c_1/x_{i_1}]_{j_1+1} \dots [c_p/x_{i_p}]_{j_p+1} \xrightarrow[B]{} b_1[d_1/x_{i_1}]_{j_1+1} \dots [d_p/x_{i_p}]_{j_p+1}$$

  and

$$\lambda x_{i_0} \cdot (a_1[c_1/x_{i_1}]_{j_1+1} \dots [c_p/x_{i_p}]_{j_p+1}) \xrightarrow[B]{} \lambda x_{i_0} \cdot (b_1[d_1/x_{i_1}]_{j_1+1} \dots [d_p/x_{i_p}]_{j_p+1})$$

  and we can apply the induction hypothesis, since the left-hand side is a χ rewrite of a.

– If $a \equiv x_{i_0}[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}$, then the B redex is in the substitution part, say in c_k.

$$x_{i_0}[c_1/x_{i_1}]_{j_1} \ldots [c_k/x_{i_k}] \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[B]{} x_{i_0}[c_1/x_{i_1}]_{j_1} \ldots [d_k/x_{i_k}] \ldots [c_p/x_{i_p}]_{j_p}.$$

If $k > 1$ and $x_{i_0}[c_1/x_{i_1}]_{j_1} \xrightarrow[\chi]{} f$ then

$$f[c_2/x_{i_2}] \ldots [c_k/x_{i_k}] \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[B]{} f[c_2/x_{i_2}] \ldots [d_k/x_{i_k}] \ldots [c_p/x_{i_p}]_{j_p}$$

and the result follows by induction. If $k = 1$ and $i_0 = i_1$ then

$$x_{i_0}[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[\chi]{} rename(c_1, i_0, j_1)[c_2/x_{i_2}]_{j_2} \ldots [c_p/x_{i_p}]_{j_p+1}$$

$$x_{i_0}[d_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[\chi]{} rename(d_1, i_0, j_1)[c_2/x_{i_2}]_{j_2} \ldots [c_p/x_{i_p}]_{j_p+1}.$$

Clearly, if $c_1 \xrightarrow[B]{} d_1$, then $rename(c_1, i_0, j_1) \xrightarrow[B]{} rename(d_1, i_0, j_1)$ and the result follows by induction. If $k = 1$ and $i_0 < i_1$ then

$$x_{i_0}[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[\chi]{} x_{i_0}[c_2/x_{i_2}]_{j_2} \ldots [c_p/x_{i_p}]_{j_p+1}$$

$$x_{i_0}[d_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p} \xrightarrow[\chi]{} x_{i_0}[c_2/x_{i_2}]_{j_2} \ldots [c_p/x_{i_p}]_{j_p+1}$$

and both normal forms by χ are equal and the results holds trivially, the same if $k = 1$ and $i_0 > i_1$ except that x_{i_0} becomes x_{i_0-1}.

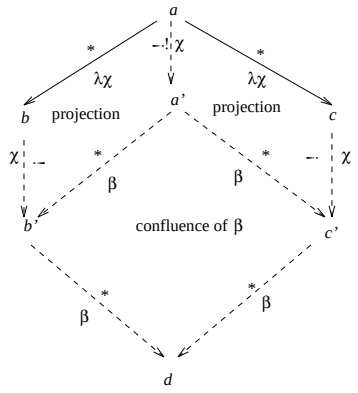
– If $a \equiv rename(a'', i, j)[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}$, then the B redex is either in a'' or in one of the c_i's. Two cases may occur, one rewrite of $rename(a'', i, j)$ preserves the B redex: $a'' = a_1\ a_2$ and the B redex occurs in a_1 or in a_2 or $a'' = \lambda x_i a_1$, then one proceeds by induction. The second case is when $a'' = (\lambda x_{i+k} a_1)\ a_2$, then $b' \equiv rename(a_1[a_2/x_{i+k}]_0, i, j)$,

$$\begin{array}{lcl}
\chi(a) & = & \chi(rename((\lambda x_{i+k} \cdot a_1)\ a_2, i, j)[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}) \\
& = & \chi((\lambda x_{i+k+j} \cdot rename(a_1, i, j))rename(a_2, i, j)[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}) \\
& \xrightarrow[B]{} & \chi(rename(a_1, i, j)[rename(a_2, i, j)/x_{i+k+j}]_0[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}) \\
& = & \chi(rename(a_1[a_2/x_{i+k}]_0, i, j)[c_1/x_{i_1}]_{j_1} \ldots [c_p/x_{i_p}]_{j_p}) \\
& \equiv & \chi(b).
\end{array}$$

The last equality follows from Lemma 2.

Theorem 1 (Confluence). *$\lambda\chi$ is confluent.*

Proof. One uses the projection lemma and Hardin's interpretation method [14] illustrated by the following picture.



6 $\lambda\chi$ preserves strong normalisation

Not all calculi of explicit substitutions preserve strong β normalisation. For instance, Paul-André Mellies [18] has exhibited strongly β normalisable terms that are not strongly $\lambda\sigma$ normalisable or strongly $\lambda\sigma_{\Uparrow}$ normalisable. On the other hand, in [17, 3], we have proved that the calculus of substitutions $\lambda\upsilon$ preserves strong normalisation. A similar proof can be used for proving that $\lambda\chi$ preserves strong normalisation. This requires a few definitions an lemmas.

Definition 2 (External position). *A position p is* internal *in a term a, if the subterm $a_{|p}$ is a subterm of c where $d[c/x_i]_j$ is a subterm of a. A position is* external *if it is not an internal position.*

Lemma 8. *If p is an external position and if $a \xrightarrow[B,p]{} b$, then $\chi(a) \xrightarrow[\beta]{+} \chi(b)$. In particular, if $\chi(a)$ is strongly normalisable, then $\chi(a) \neq \chi(b)$.*

We are going only to sketch the proof which is similar to this described in [3]. It is by contradiction based on a minimal counter-example. More precisely if we suppose that $\lambda\chi$ does not preserve strong normalisation, there exists a pure term a which is strongly β normalisable and which is not strongly $\lambda\chi$ normalisable, in other words there exists infinite $\lambda\chi$ derivations starting from a. Among those infinite derivations there exists at least one which is minimal in the following sense (see Fig. 5). At each rewrite position in the derivation one rewrites at the lowest possible position that keeps the derivation infinite, that is one may rewrite at a lower position, but the derivations which continue that rewrite are finite. The proof relies on two important lemmas.

Lemma 9 (Commutation Lemma). *If $\chi(a)$ is strongly β normalisable, $\chi(a) = \chi(b)$ and $a \xrightarrow[\lambda\chi,p]{}^{int} \cdot \xrightarrow[\chi,q]{}^{ext} b$ then $a \xrightarrow[\chi]{+}^{ext} \cdot \xrightarrow[\lambda\chi]{*}^{int} b$.*

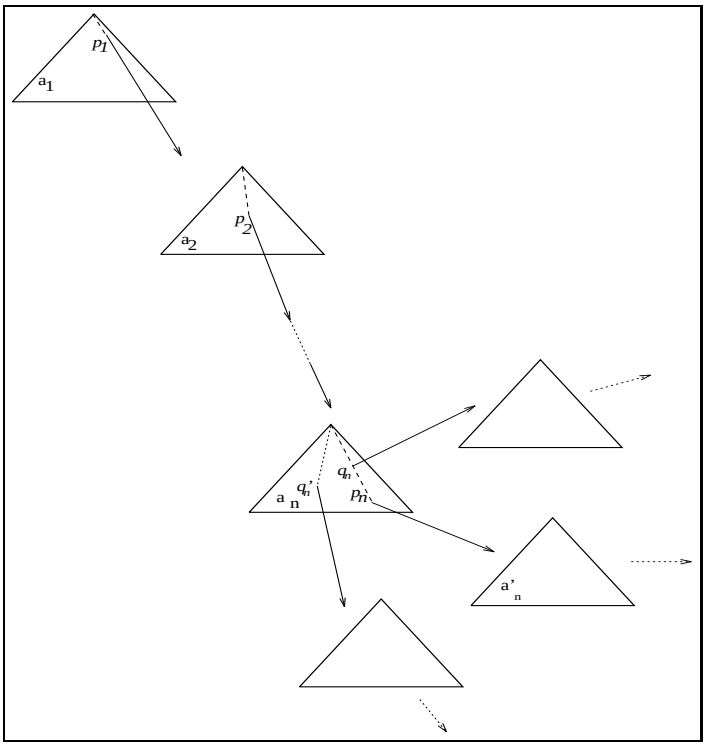


Fig. 5. A minimal $\lambda\chi$ derivation, $a_1, \ldots, a_n, a_{n+1}$

Lemma 10. *Let a_1 be a strongly β normalisable term. In each infinite $\lambda\chi$ derivation of terms*

$$a_1 \xrightarrow[\lambda\chi]{} a_2 \dots a_n \xrightarrow[\lambda\chi]{} a_{n+1} \dots$$

there exists an N such that for $i \geq N$ all the $\lambda\chi$ rewrites are internal.

Theorem 2. *Each strongly β normalisable term is strongly $\lambda\chi$ normalisable.*

Proof. The proof works as follows. If there exists an infinite $\lambda\chi$ derivation starting with a β normalisable term, one considers a minimal such derivation $\mathcal{D}$ and one uses Lemmas 8-10 to show that a B rewrite which takes place after N (see Lemma 10) can be "lifted" at $J \leq N$. The lifted rewrite on a_J is at position q lower than p_J where p_J is the position of the J^{th} $\lambda\chi$ rewrite in $\mathcal{D}$. That rewrite is continued into an infinite derivation, which contradicts the minimality of $\mathcal{D}$.

Corollary 1. *$\lambda\chi$ is strongly normalising on typed terms.*

7 Conclusion and related works

In this paper we have presented a calculus of explicit substitutions with levels. We have proved three important properties.

- $\lambda\chi$ correctly implements β conversion in the standard λ calculus.
- $\lambda\chi$ is confluent on closed (ground) χ terms, i.e., terms without variables.
- $\lambda\chi$ is preserves strong β normalisation.

Since the first presentation by de Bruijn [10], all the other calculi proposed in the literature [1, 6, 12, 13, 16, 17, 19, 20] use De Bruijn indices (except one attempt in [1] already mentioned). Our approach is therefore original by its use of levels which improves readability. We attach much importance to confluence on ground terms and to strong normalisation on typed terms. The non confluence on open terms seems to us less fundamental because of the apparent impossibility of getting both properties, e.g., confluence on open terms and preservation of strong normalisation in the same system and the inability shown by Field [13, 12] to get optimality.

Our calculus raises interesting open issues on implementation and on higher order unification. We feel indeed that due to the non superposition of left-hand sides, our $\lambda\chi$ should entail a nice description of implementations, for instance, of graph reduction based implementations. In parallel, tools for higher order theorem provers like strong normalisation and higher unification should be easily described through $\lambda\chi$.

References

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

2. L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2-3):173–202, October 1989.
3. Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. Submitted, December 1994.
4. P. Crégut. An abstract machine for the normalization of λ-calculus. In *Proc. Conf. on Lisp and Functional Programming*, pages 333–340. ACM, 1990.
5. P. Crégut. *Machines à environnement pour la réduction symbolique et l'évaluation partielle*. PhD thesis, Université de PARIS 07, 1991.
6. P.-L. Curien, Th. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. RR 1617, INRIA, Rocquencourt, February 1992.
7. H. B. Curry, R. Feys, and W. Craig. *Combinatory Logic*, volume 1. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1958.
8. N. G. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, 75(5):381–392, 1972.
9. N. G. de Bruijn. Lambda calculus with namefree formulas involving symbols that represent reference transforming mappings. *Proc. of the Koninklijke Nederlands Akademie*, 81(3):1–9, September 1978. Dedicated to A. Heyting at the occasion of his 80th Birrthday on May 9, 1978.
10. N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. TH-Report 78-WSK-03, Technological University Eindhoven, Netherlands, Department of Mathematics, 1978.
11. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.
12. J. Field. On laziness and optimality in lambda interpreters: Tools for specification and analysis. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 1–15, San Fransisco, 1990. ACM.
13. J. Field. *Incremental Reduction in the Lambda Calculus and Related Reduction Systems*. PhD thesis, Cornell U., November 1991.
14. Th. Hardin. Confluence results for the pure strong categorical combinatory logic CCL: λ-calculi as subsystems of CCL. *Theoretical Computer Science*, 65:291–342, 1989.
15. M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems. *Journal of IPS Japan*, 31(5):633–642, 1990.
16. 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.
17. P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions $\lambda\upsilon$. Technical Report RR-2222, INRIA-Lorraine, January 1994.
18. P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani, editor, *Int. Conf. on Typed Lambda Calculus and Applications*, 1995.
19. A. Ríos. *Contributions à l'étude des λ-calculs avec des substitutions explicites*. Thèse de Doctorat d'Université, U. Paris VII, 1993.
20. T. Strahm. Partial applicative theories and explicit substitutions. Technical Report IAM 93–008, Univerität Bern, Institut für Informatik und angewandte Mathematik, June 1993.