# Confluence properties of Weak and Strong Calculi of Explicit Substitutions

Pierre-Louis Curien*
Thérèse Hardin†
Jean-Jacques Lévy‡

July 24, 1991

**Abstract**

Categorical combinators [12, 21, 43] and more recently $\lambda\sigma$-calculus [1, 23], have been introduced to provide an explicit treatment of substitutions in the $\lambda$-calculus. We reintroduce here the ingredients of these calculi in a self-contained and stepwise way, with a special emphasis on confluence properties. The main new results of the paper w.r.t. [12, 21, 1, 23] are the following:

1. We present a confluent weak calculus of substitutions, where no variable clashes can be feared.
2. We solve a conjecture raised in [1]: $\lambda\sigma$-calculus is not confluent (it is confluent on ground terms only).

This unfortunate result is "repaired" by presenting a confluent version of $\lambda\sigma$-calculus, named the $\lambda Env$-calculus in [23], called here the confluent $\lambda\sigma$-calculus.

## 1 Introduction

Weak reduction in $\lambda$-calculus is defined by forbidding reduction under $\lambda$, that is by forbidding the following $\xi$-rule:

$$(\xi) \quad \frac{a \to b}{\lambda x.a \to \lambda x.b}$$

*Ecole Normale Supérieure, Paris
†INRIA, Rocquencourt, and Conservatoire National des Arts et Métiers, Paris
‡INRIA, Rocquencourt

The following rules remain:

$$(\beta) \quad (\lambda x.a)b \to a\{b/x\}$$

$$(\nu) \quad \frac{a \to a'}{ab \to a'b}$$

$$(\mu) \quad \frac{b \to b'}{ab \to ab'}$$

Most implementations of $\lambda$-calculus based languages "perform" only weak reductions: functions are considered as values, and are only evaluated when arguments are fed in. For instance, weak reduction is powerful enough to evaluate (in normal order) normal forms of closed expressions of basic types in typed functional programming languages like ML or Miranda, as we briefly explain now.

We adopt for simplicity the minimum setting for the discussion to make sense, and assume that the language has only one basic type, and one constant $\star$ of this basic type. Call *weak head normal form* a term of the form $\lambda x.a$, $xa_1 \ldots a_n$ $(n \geq 0)$ or $\star$. A term not having one of these forms is of the form $(\lambda x.a)ba_1 \ldots a_n$ $(n \geq 0)$. Its leftmost-outermost redex can be reduced by the weak rules (even without the use of $\mu$). In other words, as long as a weak head normal form has not been reached, leftmost-outermost reduction is weak. Now notice that a closed expression of basic type cannot be a weak head normal form unless it is $\star$, since $\lambda x.a$ has a functional type, and $xa_1 \ldots a_n$ is open. Hence the leftmost-outermost evaluation of a closed (and terminating) expression of basic type to $\star$ is weak. The relevance of normal reduction to weak head normal form is enhanced by the well known result, proved using the standardization theorem , that a $\lambda$-term has a (weak) head normal form iff its normal order reduction reaches a (weak) head normal form.

So far, weak reduction is nice. But it is not confluent. Take $a = (\lambda x.x)x$, $b = x$. Then

$$(\lambda y.\lambda x.y)a \to \lambda x.a$$
$$(\lambda y.\lambda x.y)a \to (\lambda y.\lambda x.y)b \to \lambda x.b$$

The point is that we have forbidden reduction under $\lambda$, preventing to further reduce $\lambda x.a$, but we have not forbidden *substitution* under $\lambda$, which has "frozen" $a$ as a side-effect of the reduction $(\lambda y.\lambda x.y)a \to \lambda x.a$. Because of the lack of confluence, only special (call-by-value and call-by-name) weak strategies have been studied in the literature (see e.g. [2, 37]).

We take here our point of departure towards explicit substitutions. In order to "implement" the idea of "no substitution under $\lambda$", we have to introduce *functional closures*, i.e. pairs consisting of an abstraction $\lambda x.a$ and a *substitution* $(b/y)$. We use (provisionally) the notation $(\lambda x.a)[b/y]$, and warn the reader that here $[\,]$ does not denote as usual a $\lambda$-term which is the result of an operation on $\lambda$-terms, but a new operator of the syntax. We reserve $\{\}$ to denote the substitution viewed as an operation from $\lambda$-terms to $\lambda$-terms. The substitution process is carried as usual, with the exception that now substitution stops before an abstraction, giving instead rise to a closure. Let us reformulate the above critical pair in this new setting:

$$(\lambda y.\lambda x.y)a \to (\lambda x.y)[a/y] \to (\lambda x.y)[b/y]$$
$$(\lambda y.\lambda x.y)a \to (\lambda y.\lambda x.y)b \to (\lambda x.y)[b/y]$$

The example suggests that confluence can be recovered in this way and it is indeed the case (see later in the introduction, and section 2).

The $\beta$-axiom and the new syntax $(\lambda x.a)[b/y]$ are not sufficient. If we start from $(\lambda y.\lambda x.y)ab$, the first step yields $((\lambda x.y)[a/y])b$, which is not a redex. This suggests to write now $\beta$ as:

$$((\lambda x.a)[s])b \to a[(b/x) \cdot s]$$

Usual $\beta$ may be recovered by introducing a dummy substitution $id$ and writing everywhere $(\lambda x.a)[id]$ in place of $\lambda x.a$. Alternatively, one may enclose the whole expression $a$ to be evaluated in a closure $a[id]$. Thus a substitution is either $id$ or a *cons*, i.e. $(a/x) \cdot s$ for some term $a$ and substitution $s$.

So far, we have a (weak) calculus with "semi-explicit" substitutions, where substitution is explicit in functional closures, but remains an operation on terms which is defined apart. (This setting is also used in [6]). It yields a confluent calculus, as will be shown in section 2. We take a natural step further, by considering substitution explicitly in the syntax, together with specific rules describing how it is performed. In other words, we decompose $\beta$ into a "formal" rule

$$\textit{(Beta')} \quad ((\lambda x.a)[s])b \to a[(b/x) \cdot s]$$

and rules describing how substitution is performed (see section 2).

Explicit (or semi-explicit) substitutions have another advantage: not only they ensure Church-Rosser in the weak setting, but also they allow

to get away with name clashes. Recall that $\beta$-reduction may force $\alpha$-conversions to prevent variable captures. The minimal example of this is: $(\lambda x \lambda y.x)y \to \lambda z.y$, where $\alpha$-conversion $\lambda y.x =_\alpha \lambda z.x$ has to be performed *before* the reduction. To be more precise, $\alpha$-conversion was needed before substitution of $y$ could be carried under $\lambda$. But we have precisely prevented such substitutions to be performed: then no clash has to be feared, and no $\alpha$-conversion has ever to be performed.

In section 2, we describe two weak calculi of explicit substitutions. The first one, which is weaker than the second, is presented by a conditional theory (in the sense that some inference rules are forbidden, just like $\xi$ above). The second one is a rewriting system in the usual sense. The removal of the restriction on inferences provokes critical pairs, which are solved at the price of adding a new operator: *composition* of substitutions. We call these two calculi *conditional weak*, and *weak* calculus, respectively.

In section 3, we address strong reduction. Here we have to move from the usual notation of $\lambda$-calculus to De Bruijn's notation, where variable names are replaced by numbers recording their binding depth. Indeed we do not yet know of any satisfactory treatment of explicit substitution keeping the usual notation of variables. The De Bruijn's notation provides an automatic way of dealing with name clashes, by means of an additional operator, $\uparrow$, and of specific rules for it. The resulting calculus is $\lambda\sigma$-calculus as considered in [1]. In [1] we proved that this calculus is ground confluent, i.e. it is confluent on the subset of ground terms. Notice that this subset includes all $\lambda$-terms, since variables of the $\lambda$-calculus act as constants in our explicit calculi. We emphasize this by calling metavariables the term and substitution variables. Also, to avoid confusion, we shall reserve "closed" for the usual notion of closed $\lambda$-term (no free variable) and "ground" for the metavariable free terms/substitutions.

$\lambda\sigma$-calculus with metavariables has not even the local confluence property. It may be recovered by adding rules, including a *surjective pairing* rule. We call $\lambda\sigma_{SP}$ the resulting theory. We prove here a negative result which was conjectured in [1]: $\lambda\sigma_{SP}$ is not confluent. We adapt the techniques of [29] and [21] to show our non confluence result. The adaptation is twofold: first we exhibit a simpler counterexample leading to a simpler proof of non confluence of $\lambda$-calculus + surjective pairing, then we "transfer" this example to $\lambda\sigma$-calculus.

In section 4, we recover full confluence: in order to prevent some critical pairs, we introduce a new operator $\Uparrow$ when substitution crosses lambda abtractions. This trick, originally in [23], not only helps for confluence, but

also simplifies proofs of termination, which are very complicated in the case of the previous $\lambda\sigma$-calculus. The new calculus, called $\lambda Env$ in [23], captures the power of the full $\lambda$-calculus and is named here the *confluent $\lambda\sigma$-calculus*.

The confluence of weak $\lambda\sigma$-calculus, the ground confluence of $\lambda\sigma$-calculus, the non-confluence of $\lambda\sigma_{SP}$-calculus are all proved using the same method, which was identified in [21], where it was used for categorical combinators[1]. We shall describe it now, since it serves as a common nerve to a large part of the paper. It can be formulated for any abstract reduction system (i.e. relation) $\mathcal{R}$.

**Lemma 1.1 (Interpretation lemma)** *Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ be the union of two relations, $\mathcal{R}_1$ being confluent and strongly normalizing. We denote by $\mathcal{R}_1(a)$ the $\mathcal{R}_1$-normal form of $a$. Suppose that there is some relation $\mathcal{R}'$ on $\mathcal{R}_1$-normal forms satisfying:*

$$\mathcal{R}' \subseteq \mathcal{R}^\star \quad \text{and} \quad (a \overset{\mathcal{R}_2}{\to} b \quad \Rightarrow \quad \mathcal{R}_1(a) \overset{\mathcal{R}'^\star}{\to} \mathcal{R}_1(b))$$

*Then $\mathcal{R}'$ is confluent iff $\mathcal{R}$ is confluent.*[2]

**Proof:** Suppose first that $\mathcal{R}'$ is confluent. Let $u \overset{\mathcal{R}^\star}{\to} u'$ and $u \overset{\mathcal{R}^\star}{\to} u''$. Then by assumption $\mathcal{R}_1(u) \overset{\mathcal{R}'^\star}{\to} \mathcal{R}_1(u')$ and $\mathcal{R}_1(u) \overset{\mathcal{R}'^\star}{\to} \mathcal{R}_1(u'')$. By confluence of $\mathcal{R}'$, there exists $v$ s.t. $\mathcal{R}_1(u') \overset{\mathcal{R}'^\star}{\to} v$ and $\mathcal{R}_1(u'') \overset{\mathcal{R}'^\star}{\to} v$. Since $\mathcal{R}' \subseteq \mathcal{R}^\star$, we have $u' \overset{\mathcal{R}^\star}{\to} \mathcal{R}_1(u') \overset{\mathcal{R}^\star}{\to} v$, and similarly $u'' \overset{\mathcal{R}^\star}{\to} v$, hence $\mathcal{R}$ is confluent.

Suppose conversely that $\mathcal{R}$ is confluent. Let $u \overset{\mathcal{R}'^\star}{\to} u'$ and $u \overset{\mathcal{R}'^\star}{\to} u''$. By confluence of $\mathcal{R}$, there exists $v$ s.t. $u' \overset{\mathcal{R}^\star}{\to} v$ and $u'' \overset{\mathcal{R}^\star}{\to} v$. Then, by assumption, we have: $u' = \mathcal{R}_1(u') \overset{\mathcal{R}'^\star}{\to} \mathcal{R}_1(v)$ and $u'' \overset{\mathcal{R}'^\star}{\to} \mathcal{R}_1(v)$, and the diagram is closed. □

Basic familiarity with $\lambda$-calculus and term rewriting systems is assumed, but it is not necessary that the reader be acquainted with previous work on explicit substitutions or categorical combinators.

## 2 Weak calculi

In this section, we shall first define the conditional weak $\lambda\sigma$-calculus which has no critical pairs, and is thus confluent (2.1). Then, in order to get a

[1] The method has been used independently elsewhere, e.g. in [7].

[2] In fact, all what is needed from $\mathcal{R}_1$ is a mapping from any $a$ to some canonical form $\mathcal{R}_1(a)$, see [21].

standard equational theory, we shall remove the conditions and solve critical pairs at the price of adding a new composition operation on substitutions. The resulting calculus is confluent too (2.3). We shall discuss a name-free version of weak $\lambda\sigma$-calculus 2.4. The computing powers of conditional weak $\lambda\sigma$-calculus and weak $\lambda\sigma$-calculus are discussed (2.2, 2.5 respectively).

## 2.1 Conditional Weak $\lambda\sigma$-calculus

We first summarize the syntax introduced in section 1, with the only difference that we accept abstractions as terms, in order to allow $\lambda$-terms to form a subset of the explicit (conditional weak) calculus.

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid x \mid ab \mid \lambda x.a \mid a[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid (a/x) \cdot s$ |

$\mathtt{X}$ and $\mathtt{x}$ are term and substitution (meta) variables. A term of the form $a[s]$ is called a *closure*. The variables $x$ of $\lambda$-calculus act here as constants, or "labels". The terms and substitutions of the above syntax are called $\lambda\sigma_{cw}$-terms, or conditional weak $\lambda\sigma$-terms. We define the *conditional weak* theory in figure 1.

It is best to point out which inference rules are *not* included in the system: we do not allow reduction under $\lambda$, nor reduction in the term part $a$ of a closure $a[s]$. The last restriction is actually the only important one, because it prevents any critical pairs. The following result also holds when reduction under $\lambda$ is allowed.

**Proposition 2.1** *$\lambda\sigma_{cw}$ is confluent.*

**Proof:** There are no critical pairs. Thus the system is orthogonal in an extended sense, which is defined and studied in [35]. The reader may want to work out a direct proof by defining a notion of parallel reduction in the most obvious way, and by showing that the diamond property holds for this parallel reduction. □

## 2.2 Computing power of conditional weak $\lambda\sigma$-calculus

Conditional weak $\lambda\sigma$-calculus is powerful enough to compute (in normal order) weak head normal forms of terms $a[id]$ whenever a $\lambda$-term $a$ has a weak head normal form. So it can be used to compute normal forms of closed expressions of basic type of, say ML, as quoted in the introduction.

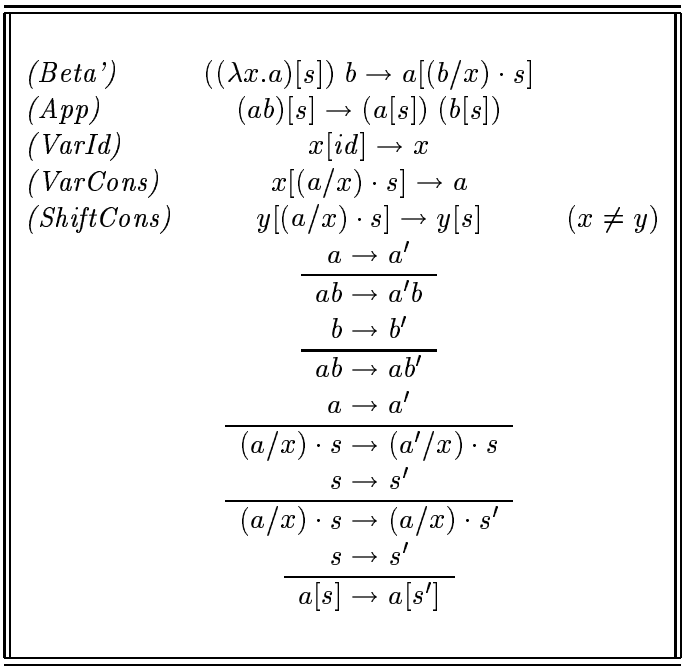



Figure 1: The conditional weak theory $\lambda\sigma_{cw}$

We shall prove this in two stages. In this subsection, we show that the conditional weak normal order strategy (defined below), if it terminates, does terminate with a weak head normal form (defined below). In 3.3 we shall show that this strategy terminates if and only if a naturally associated weak head normal strategy in classical $\lambda$-calculus terminates.

We first remark that the rules are too poor to evaluate *any* conditional weak $\lambda\sigma$-term. Suppose for example that we start with $(\lambda x.a)b$. The root is not a *Beta'*-redex, so it cannot be reduced. If we start with $((\lambda x.a)[s]\ b)[t]$, then the restriction on inferences disallows the reduction of $((\lambda x.a)[s])b$, so the only choice is the reduction of the root. The reduct $((\lambda x.a)[s][t])\ b[t]$ is *not* a *Beta'*-redex, and, worse, cannot become a *Beta'*-redex (this will be remedied to in the next subsection). These examples justify that we focus on a restricted syntax in order to make operational sense of the conditional weak $\lambda\sigma$-calculus.

We first specify normal order evaluation by the following set of rules:

$$((\lambda x.a)[s])b \overset{cwn}{\to} a[(b/x) \cdot s]$$
$$(ab)[s] \overset{cwn}{\to} (a[s])(b[s])$$
$$x[id] \overset{cwn}{\to} x$$
$$x[(a/x) \cdot s] \overset{cwn}{\to} a$$
$$y[(a/x) \cdot s] \overset{cwn}{\to} y[s]$$
$$\frac{a \overset{cwn}{\to} a'}{ab \overset{cwn}{\to} a'b}$$

Then we specify a restricted syntax (we make use of an auxiliary syntax: $\lambda$-terms!)

| | |
|---|---|
| **Terms** | $a ::= x \mid M[s] \mid a(M[s])$ |
| **Substitutions** | $s ::= id \mid (M[t]/x) \cdot s$ |
| $\lambda$ **– terms** | $M ::= x \mid MN \mid \lambda x.M$ |

In particular, if $M$ is a $\lambda$-term, then $M[id]$ is a restricted term.

**Proposition 2.2** *1. If $f$ is a term/substitution of the restricted syntax, and $f \overset{cwn}{\to} f'$, then $f'$ is also a term/substitution of the restricted syntax.*

*2. If $a$ is an $\overset{cwn}{\to}$irreducible term of the restricted syntax, then it is a* weak head normal form*, i.e. has the form $(\lambda x.M)[s]$ or $x(M_1[s_1]) \ldots (M_n[s_n])$.*

**Proof:** We prove the first part of the statement by induction on the size of $f$.

1. If $f = x[id]$, then $f' = x$ is restricted.
2. If $f = x[(a/x) \cdot s]$, then, since $f$ is restricted, $a = f'$ must be a restricted term.
3. If $f = y[(a/x) \cdot s]$, then, since $f$ is restricted, $s$ must be restricted, hence $f' = y[s]$ is restricted.
4. If $f = (MN)[s]$, then $f' = M[s]N[s]$, which is restricted by definition.
5. If $f = ((\lambda x.M')[s])M[t]$, then $f' = M'[(M[t]/x) \cdot s]$ is also restricted.
6. If $f = aM[s] \overset{cwn}{\to} a'M[s]$, then the result follows by induction.

We now prove the second part of the statement, also by induction on the size of $a$, and by cases on the structure of $a$.

1. If $a = M[s]$, then, since $a$ is irreducible, $M$ cannot be an application nor a variable. Hence $M$ is an abstraction and $a$ is a weak head normal form.

2. If $a = x$, then it is a weak head normal form.

3. If $a = b(M[s])$, then by induction $b$ is a weak head normal form. $b = (\lambda x.M)[t]$ is impossible since $a$ is irreducible. Hence $b = x(M_1[s_1])\ldots(M_n[s_n])$ for some $M_1[s_1], \ldots, M_n[s_n]$, which entails that also $a = x(M_1[s_1])\ldots(M_n[s_n])(M[s])$ is a weak head normal form.

□

## 2.3 Weak $\lambda\sigma$-calculus

Our next step is to remove the restrictions to the inference rules in the previous theory. The resulting system admits one critical pair, which we now examine:

$$(((\lambda x.a)[s])b)[t] \to a[(b/x) \cdot s][t]$$
$$(((\lambda x.a)[s])b)[t] \to ((\lambda x.a)[s][t])(b[t])$$

The rules introduced so far do not allow to solve this critical pair. We cannot even recognize a *Beta'*-redex at the end of the second reduction. This suggests to introduce a composition operation and "associativity" rules. So we now assume that if $s$, $t$ are substitutions, then $s \circ t$ is also a substitution, and we add to the theory the following rules:

$$\begin{array}{ll} \textit{(Clos)} & a[s][t] \to a[s \circ t] \\ \textit{(AssEnv)} & (s_1 \circ s_2) \circ s_3 \to s_1 \circ (s_2 \circ s_3) \end{array}$$

This allows to continue the two branches:

$$a[(b/x) \cdot s][t] \to a[((b/x) \cdot s) \circ t]$$
$$((\lambda x.a)[s][t])(b[t]) \to ((\lambda x.a)[s \circ t])(b[t]) \to a[(b[t]/x) \cdot (s \circ t)]$$

What is now missing is a distribution rule:

$$\textit{(MapEnv)} \quad ((a/x) \cdot s) \circ t \to (a[t]/x) \cdot (s \circ t)$$

Also, the superposition of *VarId* and *Clos* yields a critical pair, which can be solved if the left identity rule is added to the system:

$$\textit{(IdL)} \quad id \circ s \to s$$

| | | | | |
|---|---|---|---|---|
| *(Beta')* | $((\lambda x.a)[s])\ b$ | $\rightarrow$ | $a[(b/x) \cdot s]$ | |
| *(App)* | $(ab)[s]$ | $\rightarrow$ | $(a[s])\ (b[s])$ | |
| *(VarId)* | $x[id]$ | $\rightarrow$ | $x$ | |
| *(VarCons)* | $x[(a/x) \cdot s]$ | $\rightarrow$ | $a$ | |
| *(ShiftCons)* | $y[(a/x) \cdot s]$ | $\rightarrow$ | $y[s]$ | $x \neq y$ |
| *(Clos)* | $a[s][t]$ | $\rightarrow$ | $a[s \circ t]$ | |
| *(AssEnv)* | $(s_1 \circ s_2) \circ s_3$ | $\rightarrow$ | $s_1 \circ (s_2 \circ s_3)$ | |
| *(IdL)* | $id \circ s$ | $\rightarrow$ | $s$ | |
| *(MapEnv)* | $((a/x) \cdot s) \circ t$ | $\rightarrow$ | $(a[t]/x) \cdot (s \circ t)$ | |

Figure 2: The weak theory $\lambda\sigma_w$

Collecting all this material, we now have the following syntax:

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid x \mid ab \mid (\lambda x.a)[s] \mid a[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid (a/x) \cdot s \mid s \circ t$ |

We define the *weak* theory $\lambda\sigma_w$ in figure 2. The subtheory consisting of all the rules except *Beta'* is called $\sigma_w$. The terms and substitutions of the above syntax are called $\lambda\sigma_w$-terms, or weak $\lambda\sigma$-terms. Remark that abstractions are taken into account only with substitutions, as in the restricted syntax of section 2.2. A standard $\lambda$-term $M$ can be encoded by replacing any abstraction subterm $\lambda x.M$ by $(\lambda x.M)[id]$ (see section 2.5).

The proof of confluence is more involved than in the conditional weak system, due to the presence of critical pairs. Parallelization cannot be used rightaway (the problem is with the associativity rules, see [21]). We shall make use of the interpretation method. We divide the theory $\lambda\sigma_w$ in two parts: the rule *Beta'* on one side, and the set $\sigma_w$ of all the other rules on the other side.

The confluence proof relies on three lemmas, the two first of which have an interest of their own.

**Lemma 2.3** *$\sigma_w$ is confluent and strongly normalizing.*

**Proof:** Local confluence is easily checked. Strong normalization is established using a rpo (recursive path ordering) [18, 27], based on "[ ]" and "$\circ$"

with an equal precedence, greater than the application and "$\cdot$". $\Box$

We shall denote $\sigma_w(f)$ the $\sigma_w$-normal form of $f$, where $f$ stands for a term or a substitution (and shall proceed similarly with subsequent calculi). It is easy to check that the form of $\sigma_w$-normal forms is:

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid x \mid ab \mid (\lambda x.a)[s] \mid \mathtt{X}[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid (a/x) \cdot s \mid \mathtt{x} \circ s$ |

Next we define the following reduction $\beta_w$ between $\sigma_w$-normal forms: let $f \stackrel{\beta_w}{\rightarrow} g$ when $f \stackrel{Beta'}{\rightarrow} f'$ and $\sigma_w(f') = g$, for some $f'$.
The ground $\sigma_w$-normal forms are exactly the semi-explicit $\lambda$-terms of [6], and it is easily checked that on ground terms $\beta_w$ is exactly the weak notion of $\beta$ reduction considered by [6]. We can use the parallelization method to prove the confluence of $\beta_w$.

**Lemma 2.4** *The relation $\beta_w$ is confluent on $\sigma_w$-normal forms.*

**Proof:** We shall abbreviate $\sigma_w$ into $\sigma$ in the proof. We define the following relation $\Rightarrow$ on terms/substitutions in $\sigma$-normal form:

$$\frac{}{f \Rightarrow f}$$

$$\frac{a \Rightarrow a' \quad b \Rightarrow b' \quad s \Rightarrow s'}{((\lambda x.a)[s]\ b) \Rightarrow \sigma(a'[(b'/x) \cdot s'])}$$

$$\frac{a \Rightarrow a' \quad b \Rightarrow b'}{ab \Rightarrow a'b'}$$

$$\frac{a \Rightarrow a' \quad s \Rightarrow s'}{(\lambda x.a)[s] \Rightarrow (\lambda x.a')[s']}$$

$$\frac{s \Rightarrow s'}{\mathtt{X}[s] \Rightarrow \mathtt{X}[s']}$$

$$\frac{a \Rightarrow a' \quad s \Rightarrow s'}{(a/x) \cdot s \Rightarrow (a'/x) \cdot s'}$$

$$\frac{s \Rightarrow s'}{\mathtt{x} \circ s \Rightarrow \mathtt{x} \circ s'}$$

It is clear that $\stackrel{\beta_w}{\rightarrow} \subseteq \Rightarrow \subseteq \stackrel{\beta_w}{\rightarrow}^{\star}$. Thus it is enough to show that $\Rightarrow$ satisfies the diamond property. We have to prove that if $f \Rightarrow f'$ and $f \Rightarrow f''$, then $f' \Rightarrow g$ and $f'' \Rightarrow g$ for some $g$. We prove this by induction on the size of $f$, and by cases.

If $f = ab$, $ab \Rightarrow a'b'$ and $ab \Rightarrow a''b''$, where $a \Rightarrow a'$, $b \Rightarrow b'$, $a \Rightarrow a''$ and $b \Rightarrow b''$, then by induction there exists $a'''$ and $b'''$ s.t. $a' \Rightarrow a'''$, $a'' \Rightarrow a'''$, $b' \Rightarrow b'''$ and $b'' \Rightarrow b'''$, hence $a'b' \Rightarrow a'''b'''$ and $a''b'' \Rightarrow a'''b'''$.

If $f = (a/x) \cdot s$ or $f = (\lambda x.a)[s]$, then we reason as in the previous case. If $f = \mathtt{X}[s]$ or $f = \mathtt{x} \circ s$, then reductions take place in $s$, and the result also follows by induction.

We are left with two cases, where $f = ((\lambda x.a)[t])b$, $f' = \sigma(a'[(b'/x) \cdot t'])$ and $a \Rightarrow a'$, $b \Rightarrow b'$ and $t \Rightarrow t'$. Either $f'' = \sigma(a''[(b''/x) \cdot t''])$ or $f'' = ((\lambda x.a'')[t''])b''$, where $a \Rightarrow a''$, $b \Rightarrow b''$ and $t \Rightarrow t''$. By induction there exist $a'''$, $b'''$ and $t'''$ s.t. $a' \Rightarrow a'''$, $a'' \Rightarrow a'''$, $b' \Rightarrow b'''$, $b'' \Rightarrow b'''$, $t' \Rightarrow t'''$ and $t'' \Rightarrow t'''$. We can conclude using the following claim:

**Claim**: Let $a$, $s$ and $t$ be $\sigma$-normal forms. If $a \Rightarrow a'$ and $s \Rightarrow s'$, then $\sigma(a[s]) \Rightarrow \sigma(a'[s'])$. If $t \Rightarrow t'$ and $s \Rightarrow s'$, then $\sigma(t \circ s) \Rightarrow \sigma(t' \circ s')$.

We prove the two parts of the claim together, by induction on the size of $f$, where $f$ is either $a$ or $t$, and by cases on the last inference rule used to derive $a \Rightarrow a'$ or $t \Rightarrow t'$. We do not handle the reflexivity case $f \Rightarrow f$ explicitly: but the reader may check that the proof also deals with this case.

1. If $a = x$ and $s = (b/x) \cdot s_1$, then $a' = x$ and $s' = (b'/x) \cdot s_1'$ where $b \Rightarrow b'$ and $s_1 \Rightarrow s_1'$. Hence $\sigma(a[s]) = b \Rightarrow b' = \sigma(a'[s'])$.

2. If $a = y$ and $s = (b/x) \cdot s_1$, then $a' = y$ and $s' = (b'/x) \cdot s_1'$ where $b \Rightarrow b'$ and $s_1 \Rightarrow s_1'$. Hence $\sigma(a[s]) = y[s_1] \Rightarrow y[s_1'] = \sigma(a'[s'])$.

3. If $a = \mathtt{X}$, then $a' = \mathtt{X}$, and $\sigma(a[s]) = \mathtt{X}[s] \Rightarrow \mathtt{X}[s'] = \sigma(a'[s'])$.

4. If $a = a_1 a_2$, $a' = a_1' a_2'$, $a_1 \Rightarrow a_1'$ and $a_2 \Rightarrow a_2'$, then $\sigma(a[s]) = (\sigma(a_1[s]))(\sigma(a_2[s]))$. By induction we have both $\sigma(a_1[s]) \Rightarrow \sigma(a_1'[s'])$ and $\sigma(a_2[s]) \Rightarrow \sigma(a_2'[s'])$, whence the result follows.

5. If $a = \mathtt{X}[t_1]$, then $a' = \mathtt{X}[t_1']$ and $t_1 \Rightarrow t_1'$. We have $\sigma(a[s]) = \mathtt{X}[\sigma(t_1 \circ s)]$ and $\sigma(a'[s']) = \mathtt{X}[\sigma(t_1' \circ s')]$. By induction we have $\sigma(t_1 \circ s) \Rightarrow \sigma(t_1' \circ s')$, whence the result.

6. If $a = (\lambda x.a_1)[t_1]$, then $a' = (\lambda x.a_1')[t_1']$, $a_1 \Rightarrow a_1'$ and $t_1 \Rightarrow t_1'$. We have $\sigma(a[s]) = (\lambda x.a_1)[\sigma(t_1 \circ s)]$ and $\sigma(a'[s']) = (\lambda x.a_1')[\sigma(t_1' \circ s')]$. We conclude as in the previous case.

7. If $a = ((\lambda x.a_1)[t_1])b_1$, $a' = \sigma(a_1'[(b_1'/x) \cdot t_1'])$, $a_1 \Rightarrow a_1'$, $b_1 \Rightarrow b_1'$ and $t_1 \Rightarrow t_1'$, then $\sigma(a[s]) = ((\lambda x.a_1)[\sigma(t_1 \circ s)])\sigma(b_1[s])$ and $\sigma(a'[s']) =$

$\sigma(a'_1[(b'_1[s']/x)\cdot(t'_1 \circ s')]) = \sigma(a'_1[(\sigma(b'_1[s'])/x)\cdot\sigma(t'_1 \circ s')])$ . By induction we have $\sigma(b_1[s]) \Rightarrow \sigma(b'_1[s'])$ and $\sigma(t_1 \circ s) \Rightarrow \sigma(t'_1 \circ s')$, whence the result.

8. The case $t = \mathtt{x}$ is similar to the case $a = \mathtt{X}$.
9. If $t = id$, then $t' = id$, and $\sigma(t \circ s) = s \Rightarrow s' = \sigma(t' \circ s')$.
10. The case $t = (a_1/x) \cdot t_1$ is similar to the case $a = a_1 a_2$.
11. The case $t = \mathtt{x} \circ t_1$ is similar to the case $a = \mathtt{x}[t_1]$.

□

**Lemma 2.5** *Let $f$ be a weak $\lambda\sigma$ term/substitution. If $f \overset{\lambda\sigma_w}{\rightarrow} f'$, then $\sigma_w(f) \rightarrow^\star_{\beta_w} \sigma_w(f')$.*

**Proof:** The statement is obvious if $f \overset{\sigma}{\rightarrow} f'$. We need only to concentrate on *Beta'*. The proof is by induction on $(depth(f), size(f))$ (see Proposition 3.8). We omit the case inspection, which is similar to (and simpler than) the case inspection for the proof of Proposition 3.8. □

**Proposition 2.6** *$\lambda\sigma_w$ is confluent.*

**Proof:** By the interpretation lemma of section 1 and previous lemmas. □

## 2.4 Namefree weak $\lambda\sigma$-calculus

We define now an entirely "isomorphic" name-free version of weak $\lambda\sigma$-calculus. We could do it as well for conditional weak $\lambda\sigma$-calculus. We refer to [1, 12] for a smooth introduction to De Bruijn's notation. The idea is to replace variable names by a natural number recording their binding height. These natural numbers lend themselves to a very simple operational interpretation: they correspond to a position in an environment. After these short hints, we now present the following namefree syntax for weak $\lambda\sigma$-calculus:

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid \mathtt{n} \mid ab \mid (\lambda a)[s] \mid a[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid a \cdot s \mid s \circ t$ |

(where $\mathtt{n}$ ranges over nonzero natural numbers) and the following corresponding rewriting system:

| | |
|---|---|
| *(Beta')* | $((\lambda a)[s])\ b \to a[b \cdot s]$ |
| *(App)* | $(ab)[s] \to (a[s])\ (b[s])$ |
| *(VarId)* | $\mathtt{n}[id] \to \mathtt{n}$ |
| *(VarCons)* | $\mathtt{1}[a \cdot s] \to a$ |
| *(ShiftCons)* | $\mathtt{n+1}[a \cdot s] \to \mathtt{n}[s]$ |
| *(Clos)* | $a[s][t] \to a[s \circ t]$ |
| *(IdL)* | $id \circ s \to s$ |
| *(AssEnv)* | $(s_1 \circ s_2) \circ s_3 \to s_1 \circ (s_2 \circ s_3)$ |
| *(MapEnv)* | $(a \cdot s) \circ t \to a[t] \cdot (s \circ t)$ |

In order to formulate the "isomorphism" between weak $\lambda\sigma$-calculus and its name-free version, we need to define a translation from $\lambda\sigma$-calculus to the name-free weak calculus.

We suppose that the variables of the $\lambda$-calculus are denumerable and enumerated by the sequence $x_1 \ldots x_n \ldots$. The translation function takes a term $u$ and a finite list of variables $L$ (which we denote as a word), and yields a namefree term $a_L$. $L$ is called the *formal environment* of the translation.

The translation uses an auxiliary function $l$, defined on substitutions. It determines the "output" formal environment of a substitution, to be used in the translation of a closure or a substitution.

$$\begin{aligned} l(id) &= \epsilon \qquad \text{(the empty word)} \\ l((a/x) \cdot s) &= x l(s) \\ l(s \circ t) &= l(s)\ l(t) \end{aligned}$$

Here is the translation:

$$\begin{aligned} & x_\epsilon = \mathtt{i} \quad \text{if } x = x_i \\ & x_{xL} = \mathtt{1} \\ & \frac{x_L = \mathtt{n} \quad x \neq y}{x_{yL} = \mathtt{n+1}} \\ & (ab)_L = a_L b_L \\ & (\lambda x.a)_L = \lambda(a_{xL}) \\ & a[s]_L = a_{l(s)L}[s_L] \\ & id_L = id \\ & ((a/x) \cdot s)_L = a_L \cdot s_L \\ & (s \circ t)_L = s_{l(t)L} \circ t_L \end{aligned}$$

We have purposedly translated $\lambda a$ rather than the legal weak terms $(\lambda a)[s]$, the translation of which can be recovered by first translating $\lambda a$, and then $s$. In this way, our translation is ready for use in the strong setting

of section 3. We have translated only ground terms, but the translation extends straightforwardly to all terms by setting $\mathtt{X}_L = \mathtt{X}$ and $\mathtt{x}_L = \mathtt{x}$. As an example, the reader may check that $(\lambda x_1.x_1x_2)_\epsilon$ is $\lambda(13)$.

The total correspondence between weak $\lambda\sigma$-calculus and namefree weak $\lambda\sigma$-calculus is given by the next proposition.

**Proposition 2.7** *Let $f$ be a weak $\lambda\sigma$-term/substitution. If $f$ weakly reduces to $g$, then $f_L$ weakly reduces to $g_L$. If $f_L$ weakly reduces to $h$, then there exists a unique weak $\lambda\sigma$-term $g$ s.t. $f$ weakly reduces to $g$ and $h = g_L$.*

**Proof:** We do not give all the details, but choose sample cases.

1. If $f = ((\lambda x.a)[s])b$ and $g = a[(b/x) \cdot s]$, then $f_L = (\lambda a_{xl(s)L})[s_L]b_L \rightarrow a_{xl(s)L}[b_L \cdot s_L] = g_L$.
2. If $f = ((a/x) \cdot s) \circ t$ and $g = (a[t]/x) \cdot (s \circ t)$, then $f_L = (a_{l(t)L} \cdot s_{l(t)L}) \circ t_L \rightarrow a_{l(t)L}[t_L] \cdot (s_{l(t)L} \circ t_L) = g_L$.
3. If $f = y[(a/x) \cdot s]$ and $g = y[s]$, then $f_L = y_{xl(s)L}[a_L \cdot s_L] \rightarrow y_{l(s)L}[s_L] = g_L$. Conversely, if $f$ is such that $f_L = \mathtt{1}[a \cdot s]$, then $f$ must be $x[(a'/y) \cdot s']$ for some $a'$, $s'$. But $x \neq y$ is impossible as otherwise $f_L$ would have the form $\mathtt{n+1}[\ldots]$. Hence $f \rightarrow a'$, and $a'_L = a$.

□

## 2.5 Computing power of weak $\lambda\sigma$-calculus

This subsection parallels subsection 2.2. By the results of the previous subsection, we can freely move to a namefree notation.

Weak normal $\lambda\sigma$ reduction $\stackrel{wn}{\rightarrow}$ is defined by:

$$((\lambda a)[s])b \stackrel{wn}{\rightarrow} a[b \cdot s]$$
$$\frac{a \stackrel{wn}{\rightarrow} a'}{ab \stackrel{wn}{\rightarrow} a'b}$$
$$\mathtt{n}[id] \stackrel{wn}{\rightarrow} \mathtt{n}$$
$$\mathtt{1}[a \cdot s] \stackrel{wn}{\rightarrow} a$$
$$\mathtt{n{+}1}[a \cdot s] \stackrel{wn}{\rightarrow} \mathtt{n}[s]$$
$$\frac{s \stackrel{wn}{\rightarrow} s'}{\mathtt{n}[s] \stackrel{wn}{\rightarrow} \mathtt{n}[s']}$$

$$(ab)[s] \stackrel{wn}{\rightarrow} (a[s])(b[s])$$
$$a[s][t] \stackrel{wn}{\rightarrow} a[s \circ t]$$
$$(a \cdot s) \circ t \stackrel{wn}{\rightarrow} a[t] \cdot (s \circ t)$$
$$(s \circ s') \circ s'' \stackrel{wn}{\rightarrow} s \circ (s' \circ s'')$$

We adapt from Proposition 2.2 the definition of weak head normal. A term in weak head normal form is a term of the form $(\lambda a)[s]$ or $\mathtt{n}a_1 \ldots a_m$. A substitution is in weak head normal form if it is either *id* or a *cons*, i.e. if it is not a composition. We first show that reductions may stop only on weak head normal forms.

**Proposition 2.8** *Let $f$ be a weak $\lambda\sigma$-term, not in weak head normal form. Then there is always some $f'$ such that $f \stackrel{wn}{\rightarrow} f'$.*

**Proof:** We prove the statement by induction on the size of $f$, and by cases on the definition of $\stackrel{wn}{\rightarrow}$. Clearly, we need to consider only the inference cases.

1. If $f = ab$ is not a *Beta'*-redex, then $a$ cannot be a weak head normal form, since $f$ is not a weak head normal form. The result follows by induction.

2. If $f = \mathtt{1}[s]$ is neither a *VarId* nor a *VarCons* redex, then $s$ is not a weak head normal form, and the result follows by induction.

Of course this proof works because each term and substitution which is not in weak head normal form matches one (and only one) of the rules defining $\stackrel{wn}{\rightarrow}$. □

In order to evaluate a $\lambda$-term (in De Bruijn notation), one has to replace all abstraction subterms $\lambda a$ by $(\lambda a)[id]$. We leave the reader check that for

all ground $\lambda\sigma$-terms $a$, $a =_\sigma a[id]$ is provable (cf. claim in Lemma 3.2 and second claim in the proof of Proposition 3.3).

Thus we have slightly different ways of encoding a $\lambda$-term $a$ to run $\overset{cwn}{\rightarrow}$ or $\overset{wn}{\rightarrow}$: for $\overset{cwn}{\rightarrow}$, we start from $a[id]$, for $\overset{wn}{\rightarrow}$ we start from $a'$ built from $a$ as described in the previous paragraph. These differences are inessential. The reader may check that the only uses of *Clos* in a reduction from $a'$ will be on terms of the form $(\lambda a)[id][s]$.

The interest of $\overset{wn}{\rightarrow}$ is to allow for code optimizations: one can imagine, *before* running a normal order evaluation, to reduce some parts $a$ of the code in closures $a[s]$. Take for instance $a = ((\lambda a_1)[id])a_2$. Then we can first reduce $a$ to $a_1[a_2 \cdot id]$, and run $\overset{wn}{\rightarrow}$ on $a_1[a_2 \cdot id][s]$. Thus the "full power" of *Clos* and $\overset{wn}{\rightarrow}$ will be used.

# 3 $\lambda\sigma$-calculus

We complete the description of (namefree) $\lambda\sigma$-calculus by adding a new operator $\uparrow$ which allows to handle substitutions under $\lambda$ (3.1), and recall basic properties of $\lambda\sigma$-calculus established in [1]. We then connect $\lambda\sigma$-calculus with classical $\beta$ reduction and $\alpha$-conversion (3.2). We come back to weak reduction and prove the completeness of the normal order strategies investigated in 2.2 and 2.5 (3.3). Finally we prove a non confluence result (3.4).

## 3.1 A ground confluent strong calculus

De Bruijn's notation allows a mechanical treatment of $\alpha$-conversion. One introduces a new operation on top of the namefree calculus of 2.4, namely $\uparrow$, which is a constant substitution, introduced by the following rule:

$$(Abs) \quad (\lambda a)[s] \rightarrow \lambda(a[1 \cdot (s \circ \uparrow)])$$

We refer to [8] for the original discussion, and to [1] for quite extensive informal operational explanations. We briefly justify it here from a different perspective through typechecking. In [1], type systems for explicit substitutions are introduced. Terms are typed as usual, and substitutions have sequences of types as types (cf. the function $l(s)$ above). The above left hand side is well typed when $s$ has type $\Gamma$ and $\lambda a$ has type $A \rightarrow B$ in context $\Gamma$ (the context assigns types to free variables). When substitution crosses $\lambda$, it goes into the context of $a$, which is $A, \Gamma$. Let $\Delta$ be the context

| | | | |
|---|---|---|---|
| *(Beta)* | $(\lambda a)\ b$ | $\rightarrow$ | $a[b \cdot id]$ |
| *(App)* | $(ab)[s]$ | $\rightarrow$ | $(a[s])\ (b[s])$ |
| *(VarId)* | $\mathtt{1}[id]$ | $\rightarrow$ | $\mathtt{1}$ |
| *(VarCons)* | $\mathtt{1}[a \cdot s]$ | $\rightarrow$ | $a$ |
| *(Clos)* | $a[s][t]$ | $\rightarrow$ | $a[s \circ t]$ |
| *(Abs)* | $(\lambda a)[s]$ | $\rightarrow$ | $\lambda(a[\mathtt{1} \cdot (s \circ \uparrow)])$ |
| *(IdL)* | $id \circ s$ | $\rightarrow$ | $s$ |
| *(ShiftId)* | $\uparrow \circ id$ | $\rightarrow$ | $\uparrow$ |
| *(ShiftCons)* | $\uparrow \circ (a \cdot s)$ | $\rightarrow$ | $s$ |
| *(AssEnv)* | $(s_1 \circ s_2) \circ s_3$ | $\rightarrow$ | $s_1 \circ (s_2 \circ s_3)$ |
| *(MapEnv)* | $(a \cdot s) \circ t$ | $\rightarrow$ | $a[t] \cdot (s \circ t)$ |

Figure 3: The theory $\lambda\sigma$

of $s$, and let $s'$ be $s$ after it has crossed $\lambda$. Then we have to construct $s'$ as "going from $A, \Delta$ to $A, \Gamma$". We have that $\mathtt{1}$ "goes from $A, \Delta$ to $A$" (the type of the first variable of the context). $\uparrow$ becomes less mysterious when we call it second projection, "going from $A, \Delta$ to $\Delta$". The composition of $s$ and $\uparrow$ "goes from $A, \Delta$ to $\Gamma$". Finally, putting things together, we have that $s' = \mathtt{1} \cdot (s \circ \uparrow)$ "goes from $A, \Delta$ to $A, \Gamma$".

Once $\uparrow$ has been introduced, there is no more need for an infinite provision of natural numbers, we can encode $\mathtt{2}$ by $\mathtt{1}[\uparrow]$, etc... We shall assume these encodings implicitly in the rest of the section (in particular we do not change the name of *ShiftCons*).

Now we present the syntax of $\lambda\sigma$-calculus:

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid \mathtt{1} \mid ab \mid \lambda a \mid a[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid \uparrow \mid a \cdot s \mid s \circ t$ |

The theory $\lambda\sigma$ is defined in figure 3. The subtheory consisting of all the rules except *Beta* is called $\sigma$. The terms and substitutions of the above syntax are called $\lambda\sigma$-terms.

The system $\sigma$ is confluent and strongly normalizing. The proof of strong normalization actually goes back to [22], in the setting of categorical combinators. Recently, a different proof was found [15]. The ground terms in

normal form are the classical $\lambda$-terms (modulo the translation to De Bruijn's calculus, see next subsection; see also Proposition 4.10).

The theory $\lambda\sigma$ is ground confluent [1]. The method used is the interpretation method (cf. section 1). We shall investigate the problem of confluence without the restriction to ground terms in 3.4, and in section 4. For the time being, we just point at where the restriction to ground terms appears crucial for the confluence of $\lambda\sigma$-calculus.

One of the three key pieces of the interpretation method consists in proving $\sigma(a) \stackrel{\beta}{\rightarrow}{}^{\star} \sigma(b)$ whenever $a \stackrel{Beta}{\rightarrow} b$. Here $\beta$ is defined on $\sigma$-normal forms as one *Beta* step followed by $\sigma$-normalization. $\beta$ is well named, since for ground terms it coincides with classical $\beta$ reduction, as we shall see in the next subsection.

Consider the case where $a = ((\lambda a_1)a_2)[s]$ and $b = a_1[a_2 \cdot id][s]$. Easy calculations yield: $\sigma(a) \stackrel{\beta}{\rightarrow} \sigma(a_1[a_2[s] \cdot (s \circ id)])$. But, for ground terms, one can prove $\sigma(s \circ id) = \sigma(s)$ (cf. claim in Lemma 3.2 and second claim in the proof of Proposition 3.3). So the right hand side is $\sigma(b)$.

In contrast, when substitution metavariables are present, $\sigma(\mathtt{x} \circ id) = \mathtt{x} \circ id \neq_\sigma \mathtt{x} = \sigma(\mathtt{x})$, hence this rule must be explicitly added to the system. This discussion will be pursued in 3.4.

## 3.2 $\alpha$-conversion and $\beta$-reduction

We pause a moment and prove in detail that $\lambda\sigma$-calculus really implements $\beta$, and that De Bruijn's notation acts as a quotient w.r.t. $\alpha$-conversion.

Let us define the set of $\lambda$-terms in De Bruijn notation (De Bruijn terms for short) as the subset of terms generated by the clauses

**Terms** $\quad a ::= \mathtt{n} \mid ab \mid \lambda a$

We need a closer look at the explicit definition of $\beta$ given at the end of last section. We defined $\beta$ as one *Beta* step followed by $\sigma$-normalization. It is not immediate from this definition that $\beta$ is a congruence: this needs a proof.

**Lemma 3.1** *The two following relations $\rightarrow$ and $\rightarrow'$ on De Bruijn terms are the same:*

1. *$a \rightarrow b$ iff $a \stackrel{Beta}{\rightarrow} b'$ and $b = \sigma(b')$ for some b'.*
2. *$a \rightarrow' b$ is defined by the following inference system:*

$$(\lambda a)b \to' \sigma(a[b/x])$$

$$\frac{a \to' a'}{ab \to' a'b}$$

$$\frac{b \to' b'}{ab \to' ab'} \qquad \frac{a \to' a'}{\lambda a \to' \lambda a'}$$

*We shall denote (provisionally) this relation with $\beta_\sigma$.*

**Proof:** The proof is by induction on the structure of $a$. Suppose $a = a_1 a_2$ and $a \to a'$. If the *Beta* step takes place at the root, then $a \to' a'$ by definition. If the *Beta* step takes place in $a_1$, then we have $a \stackrel{Beta}{\to} a'_1 a_2$, with $\sigma(a'_1 a_2) = a'$. We have $\sigma(a'_1 a_2) = \sigma(a'_1)\ a_2$, since $a_2$ is a $\sigma$-normal form. And by induction, $a_1 \to \sigma(a'_1)$ implies $a_1 \to' \sigma(a'_1)$, which in turn by definition implies $a_1 a_2 \to' \sigma(a'_1)a_2 = a'$. The converse direction, and the abstraction case are shown in the same way. □

We have already defined in 2.4 a translation from named syntax to unnamed syntax, which a fortiori translates $\lambda$-terms to De Bruijn terms.

We shall need a technical lemma. We write $\Uparrow(s)$ as a shorthand for $1 \cdot (s \circ \uparrow)$ (this abbreviation will become a fruitful new operation in the next section), and $\Uparrow^2(s)$ for $\Uparrow(\Uparrow(s))$, etc...

**Lemma 3.2** *Let $a$ be a $\lambda$-term. If $L'$ has length $n$ and $z \notin FV(a)$, then $a_{L'zL} =_\sigma a_{L'L}[\Uparrow^n (\uparrow)]$.*

**Proof:** The proof is by induction on the structure of $a$. The variable case is handled by easy $\sigma$-calculations. The application case follows by application of the rule *App*. The abstraction case $a = \lambda x.a'$ is almost as easy. We have $(\lambda x.a')_{L'zL} = \lambda a'_{xL'zL}$. By induction we have: $a'_{xL'zL} =_\sigma a'_{xL'L}[\Uparrow^{n+1} (\uparrow)]$. Thus, by *Abs*, we have

$$(\lambda x.a')_{L'zL} =_\sigma (\lambda a'_{xL'L})[\Uparrow^n (\uparrow)] = \lambda a_{L'L}[\Uparrow^n (\uparrow)]\ .$$

This concludes the proof. □

The previous proposition, appropriately formulated, holds more widely for all ground $\lambda\sigma$-terms. In the following proposition $\beta$ denotes the classical $\beta$-reduction [3] and $\beta_\sigma$ the reduction defined in lemma 3.1.

**Proposition 3.3** *Let $c$ be a $\lambda$-term. Then $c \stackrel{\beta}{\to} c'$ iff $c_L \stackrel{\beta_\sigma}{\to} c'_L$ (for any $L$). If $a$ and $b$ are $\alpha$-interconvertible, then, for any $L$, $a_L$ and $b_L$ coincide.*

**Proof:** For the first part of the statement (we limit ourselves to the only if part), thanks to the lemma 3.1, it is enough to check the axiom case. We have: $c_L = ((\lambda x.a)b)_L = (\lambda a_{xL})b_L \stackrel{Beta}{\rightarrow} a_{xL}[b_L \cdot id]$. Thus we have to prove the following

**Claim**: For any $\lambda$-terms $a$ and $b$, and any formal environment $L$, $a\{b/x\}_L = \sigma(a_{xL}[b_L \cdot id])$.

We prove the claim by induction on the size of $a$. Notice that it is enough to show that $a_{xL}[b_L \cdot id] =_\sigma a\{b/x\}_L$, since $\sigma$ is confluent and $a\{b/x\}_L$ is in $\sigma$-normal form.

Of course, the interesting case is: $a = \lambda y.a'$. We shall omit the other cases, which are handled like in proposition 2.7. Recall the (quite involved) definition of substitution in this case.

$$(\lambda y.a')\{b/x\} = \lambda z.(a'\{z/y\}\{b/x\}) \quad (z \neq x \quad \text{and } z \notin FV(a') \cup FV(b))$$

We have:

$$(\lambda z.(a'\{z/y\}\{b/x\}))_L = \lambda(a'\{z/y\}\{b/x\}_{zL})$$

and

$$(\lambda y.a')_{xL}[b_L \cdot id] = (\lambda a'_{yxL})[b_L \cdot id] = \lambda(a'_{yxL}[\mathtt{1} \cdot (b_L \cdot id) \circ \uparrow]) \ .$$

Observing that $a'\{z/y\}$ has the same size as $a'$, we can apply induction and obtain

$$a'\{z/y\}\{b/x\}_{zL} =_\sigma a'\{z/y\}_{xzL}[b_{zL} \cdot id] \ .$$

Applying induction once more, we obtain

$$a'\{z/y\}_{xzL} =_\sigma a'_{yxzL}[z_{xzL} \cdot id] \ .$$

Putting together, and replacing $z_{xzL}$ by its value, we obtain that $a'\{z/y\}\{b/x\}_{zL}$ is $\sigma$-equal to $a'_{yxzL}[\mathtt{2} \cdot id][b_{zL} \cdot id]$. We shall now apply lemma 3.2 twice, to replace $a'_{yxzL}$ by $a'_{yxL} \circ \Uparrow (\Uparrow (\uparrow))$ and $b_{zL}$ by $b_L[\uparrow]$. We are reduced to check the following equality:

$$\Uparrow (\Uparrow (\uparrow)) \circ (\mathtt{2} \cdot id) \circ ((b_L[\uparrow]) \cdot id) = \mathtt{1} \cdot (b_L[\uparrow] \cdot \uparrow)$$

which follows by easy computations.

Finally, we consider the $\alpha$-axiom $\lambda x.a = \lambda y.a\{y/x\}$, where $y \notin FV(a)$. Using the claim, we have:

$$(\lambda y.a\{y/x\})_L = \lambda(a\{y/x\}_{yL}) =_\sigma \lambda(a_{xyL}[y_{yL} \cdot id]) \ .$$

Since $y$ is not free in $a$, we get: $a_{xyL}[y_{yL} \cdot id] =_\sigma a_{xL}[\Uparrow (\uparrow)][\mathtt{1} \cdot id]$. We have: $\Uparrow (\uparrow) \circ (\mathtt{1} \cdot id) =_\sigma \mathtt{1} \cdot id$, so we are reduced to the following claim, which we formulate with a suitable induction load:

**Claim**: If $a$ is a De Bruijn term , then $a[\Uparrow^n (\mathtt{1} \cdot \uparrow)] =_\sigma a$ (for any $n$).

Notice that the claim can be formulated and proved more generally for all ground $\lambda\sigma$-terms. We omit the proof, which is similar to the proof of 3.2. Using the claim, we get

$$(\lambda y.a\{y/x\})_L =_\sigma \lambda(a_{xL}[\mathtt{1} \cdot id]) =_\sigma \lambda(a_{xL}) = (\lambda x.a)_L$$

which ends the proof. □

We need one more auxiliary function for our next statement. It computes the maximal level of bindings of a De Bruijn's term:

$$\delta(\mathtt{n}) = n$$
$$\delta(ab) = max(\delta(a), \delta(b))$$
$$\delta(\lambda a) = \delta(a) - 1 \text{ if } \delta(a) \neq 0 \ , \ 0 \text{ otherwise}$$

**Proposition 3.4** *1. Given any De Bruijn term $a$ and any formal environment $L$ of length at least $\delta(a)$, and consisting of distinct names, there exists a $\lambda$-term $b$ s.t. $b_L = a$.*

*2. Given a formal environment $L$, two $\lambda$-terms $b'$ and $b''$ have equal translations $a = b'_L = b''_L$ iff $b'$ and $b''$ are $\alpha$-interconvertible.*

**Proof:** We prove the statement by induction on the structure of $a$. We have already proved the "if" part of the second part of the statement.

1. If $a = \mathtt{n}$, then take $b = x$, where $x$ is the $n$-th element of $L$. Moreover $b$ is the only term s.t. $b_L = a$, hence the second part of the statement holds vacuously.

2. If $a = a_1 a_2$, then $b$, $a'$ and $a''$ must all be applications, and the result follows immediately by induction, remarking that if $L$ has length at least $\delta(a_1 a_2)$, then it has a fortiori length at least $\delta(a_1)$ and $\delta(a_2)$.

3. If $a = \lambda\overline{a}$ and $L$ satisfies the conditions of the statement relatively to $a$, then we choose $x$ not occurring in $L$. By induction, there exists $b'$ s.t. $b'_{xL} = \overline{a}$, whence we get $a = (\lambda x.b')_L$. We now prove the second part of the statement for this case. If $b'_L = \lambda\overline{a}$, then $b' = \lambda x.\overline{b'}$, for

some $x$ and $\overline{b}'$, and similarly $b'' = \lambda y.\overline{b}''$, for some $y$ and $\overline{b}''$. Hence ${\overline{b}'}_{xL} = \overline{a}$ and ${\overline{b}''}_{yL} = \overline{a}$. We choose a variable $z$ not occurring free in $b'$ nor in $b''$. Then $c' = \lambda z.\overline{b}'\{z/x\}$ is such that $c'_L = a$ by Proposition 3.3. Similarly, $c''_L = a$, where $c'' = \lambda z.\overline{b}''\{z/y\}$. Set $\overline{c}' = \overline{b}'\{z/x\}$ and $\overline{c}'' = \overline{b}''\{z/y\}$. Then, by induction, $c' =_\alpha c''$, since ${\overline{c}'}_{zL} = {\overline{c}''}_{zL} = \overline{a}$. Putting the pieces together, we get: $b' =_\alpha c' =_\alpha c'' =_\alpha b''$.

□

### 3.3 Completeness of weak normal strategies

In this subsection, we complete the results of 2.2 and 2.5 by showing that the weak computation strategies of these subsections reach weak head normal forms whenever such weak head normal forms exist in the sense of classical $\lambda$-calculus. We first recall the (weak) normal order strategy in $\lambda$-calculus.

$$(\lambda a)b \overset{wn\beta}{\rightarrow} \sigma(a[b \cdot id])$$

$$\frac{a \overset{wn\beta}{\rightarrow} a'}{ab \overset{wn\beta}{\rightarrow} a'b}$$

**Proposition 3.5** *For any weak (conditional weak) $\lambda\sigma$-term $a$, the $\overset{wn\beta}{\rightarrow}$ evaluation of $\sigma(a)$ terminates (with $b$) iff the $\overset{wn}{\rightarrow}$ evaluation ($\overset{cwn}{\rightarrow}$ evaluation) of $a$ terminates (with $a'$ s.t. $\sigma(a') = b$).*

**Proof:** We prove the statement for $\overset{wn}{\rightarrow}$, the proof being similar for $\overset{cwn}{\rightarrow}$. The statement follows from the following claim, the precise formulation and the proof of which we leave to the reader:

**Claim**: If $a \overset{wn}{\rightarrow} b$, then $\sigma(a) = \sigma(b)$ if the underlying redex is a $\sigma$-redex, and $\sigma(a) \overset{wn\beta}{\rightarrow} \sigma(b)$ if the underlying redex is a *Beta'*-redex.

It follows from the claim, and from the strong termination of $\sigma$, that any infinite $\overset{wn}{\rightarrow}$ reduction from $a$ would result in an infinite $\overset{wn\beta}{\rightarrow}$ reduction from $\sigma(a)$, which ends the proof. □

Putting together proposition 3.5 and proposition 2.2 (respectively 2.8), we obtain:

**Theorem 3.6** *For any weak (conditional weak) $\lambda\sigma$-term $a$, the $\stackrel{wn\beta}{\rightarrow}$ evaluation of $\sigma(a)$ terminates with a weak head normal form $b$ iff the $\stackrel{wn}{\rightarrow}$ evaluation ( $\stackrel{cwn}{\rightarrow}$ evaluation) of $a$ terminates with a weak head normal form $a'$ s.t. $\sigma(a') = b$.*

## 3.4 Non Confluence result

The theory $\lambda\sigma$ is ground confluent, but not even locally confluent. The discussion at the end of 3.2 has shown need for a right identity rule to solve the critical pair arising at $((\lambda a_1)a_2)[s]$. Once the rule

$$\textit{(IdR)} \quad s \circ id \rightarrow s$$

has been added, the critical pair at $(a \cdot s) \circ id$ leads us to add the other right identity rule:

$$\textit{(Id)} \quad a[id] \rightarrow a$$

And here comes the "harmful" critical pair at $(\lambda a)[id]$, which suggests to add:

$$\textit{(VarShift)} \quad \mathtt{1} \cdot \uparrow \rightarrow id$$

from which we arrive to surjective pairing, because of the critical pair at $(\mathtt{1} \cdot \uparrow) \circ s$:

$$\textit{(SCons)} \quad \mathtt{1[s]} \cdot (\uparrow \circ s) \rightarrow s$$

We obtain a locally confluent theory, which we shall call $\lambda\sigma_{SP}$, and which is described in figure 4.

We devote the rest of this subsection to show that $\lambda\sigma_{SP}$ is not confluent (it can be shown that it is still ground confluent). We use once more the interpretation method, to reduce the non-confluence in the $\lambda\sigma$-calculus to the non-confluence of a reduction system on $\sigma$-normal forms which behaves like $\lambda$-calculus extended with surjective pairing. We need to extend lemma 3.1.

**Lemma 3.7** *The following properties hold:*

1. *The set of $\sigma$-normal forms is closed under Id, IdR, VarShift and SCons reductions.*

| | | | |
|---|---|---|---|
| *(Beta)* | $(\lambda a)\ b$ | $\rightarrow$ | $a[b \cdot id]$ |
| *(App)* | $(ab)[s]$ | $\rightarrow$ | $(a[s])\ (b[s])$ |
| *(VarCons)* | $\mathtt{1}[a \cdot s]$ | $\rightarrow$ | $a$ |
| *(Clos)* | $a[s][t]$ | $\rightarrow$ | $a[s \circ t]$ |
| *(Abs)* | $(\lambda a)[s]$ | $\rightarrow$ | $\lambda(a[\mathtt{1} \cdot (s \circ \uparrow)])$ |
| *(IdL)* | $id \circ s$ | $\rightarrow$ | $s$ |
| *(ShiftCons)* | $\uparrow \circ (a \cdot s)$ | $\rightarrow$ | $s$ |
| *(AssEnv)* | $(s_1 \circ s_2) \circ s_3$ | $\rightarrow$ | $s_1 \circ (s_2 \circ s_3)$ |
| *(MapEnv)* | $(a \cdot s) \circ t$ | $\rightarrow$ | $a[t] \cdot (s \circ t)$ |
| *(Id)* | $a[id]$ | $\rightarrow$ | $a$ |
| *(IdR)* | $s \circ id$ | $\rightarrow$ | $s$ |
| *(VarShift)* | $\mathtt{1} \cdot \uparrow$ | $\rightarrow$ | $id$ |
| *(SCons)* | $\mathtt{1}[\mathtt{s}] \cdot (\uparrow \circ s)$ | $\rightarrow$ | $s$ |

Figure 4: The theory $\lambda\sigma_{SP}$

2. *The statement of Lemma 3.1 holds for any term/substitution, extending the definition of $\rightarrow'$ accordingly.*

**Proof:** The proof is by induction on the structure of $f$. We carry an additional load in the induction: if $f \rightarrow g$ where $\rightarrow$ is a *Beta*, *Id*, *IdR*, *VarShift* or *SCons* step, and if $f$ is not a *cons*, then $g$ is not a *cons*. We consider only the new cases w.r.t. lemma 3.1. We do not give all the details.

1. $f = a \cdot s$ (and similarly $f = a[s]$). The additional load holds vacuously. For the second part of the statement, we apply quite simply induction on $a$ or $s$. For the first part of the statement, we consider three cases:
   (a) The reduction takes place in $a$ or $s$. Apply induction.
   (b) $a = \mathtt{1}$ and $s = \uparrow$. The reduct $id$ is a $\sigma$-normal form.
   (c) $a = \mathtt{1}[t]$ and $s = \uparrow \circ t$. The reduct is a subterm of $f$, hence a $\sigma$-normal form.
2. $f$ is a composition. Since it is a $\sigma$-normal form, it can only have one of the two following forms:

(a) $f = \uparrow \circ s$. The only case where the root could be rewritten is $s = id$, but this is impossible since $f$ is a $\sigma$-normal form (the instance *ShiftId* of *IdR* is part of $\sigma$). Hence the reduct of $f$ has the form $\uparrow \circ s'$, where $s \rightarrow s'$. By induction, $s'$ is a $\sigma$-normal form which is not a *cons*, since $s$ is not a *cons*. So the only possibility left for the reduct not to be a $\sigma$-normal form is: $s' = id$. This could only happen either if $s$ is a *VarShift* or *SCons* redex, which is impossible since $s$ is not a *cons*, or if $s = id \circ id$, which is impossible since $id \circ id$ is also an *IdL* redex. Thus $\uparrow \circ s'$ is a $\sigma$-normal form, and is not a *cons*.

(b) $f = \mathtt{x} \circ s$. There are two cases:

i. $s = id$: The reduct $\mathtt{x}$ is a $\sigma$-normal form and is not a *cons*.

ii. $s \rightarrow s'$: One concludes by induction.

□

**Proposition 3.8** *Let $f$ be a $\lambda\sigma$-term/substitution.*

1. *If* $f \overset{Beta}{\rightarrow} g$, *then* $\sigma(f) \overset{\beta, IdR, Id^\star}{\rightarrow} \sigma(g)$ .
2. *If* $f \overset{Id, IdR}{\rightarrow} g$, *then* $\sigma(f) \overset{IdR, Id, VarShift^\star}{\rightarrow} \sigma(g)$ .
3. *If* $f \overset{VarShift}{\rightarrow} g$, *then* $\sigma(f) \overset{VarShift^\star}{\rightarrow} \sigma(g)$ .
4. *If* $f \overset{SCons}{\rightarrow} g$, *then* $\sigma(f) \overset{VarShift, SCons^\star}{\rightarrow} \sigma(g)$ .

**Proof:** We show this by induction on $(depth(f), size(f))$, where the depth of $f$ is the maximal length of a $\sigma$ derivation from $f$, and the size is the number of nodes in the tree representation of $f$. We proceed by cases on the form of $f$. We prove all the four parts of the statement simultaneously. We shall content ourselves with writing $f \rightarrow g$, $\sigma(f) \overset{\star}{\rightarrow} \sigma(g)$ when our argument is the same for all four cases.

1. $f = 1, \mathtt{X}, id, \uparrow,$ or $\mathtt{x}$. No reduction can apply to $f$, thus the statement holds vacuously.
2. $f = \lambda a$. Then $g = \lambda b$ for some $b$, and $a \rightarrow b$. We can apply induction to $a$, and thus have $\sigma(a) \overset{\star}{\rightarrow} \sigma(b)$. But $\sigma(\lambda a) = \lambda(\sigma(a))$, so we conclude that also $\sigma(\lambda a) \overset{\star}{\rightarrow} \sigma(\lambda b)$.

3. $f = ab$. There are three cases:

   (a) The reduction $f \to g$ takes place in $a$. Thus $g = a'b$, and $a \to a'$. Applying induction to $a$, we get $\sigma(a) \xrightarrow{\star} \sigma(a')$. But $\sigma(ab) = \sigma(a)\sigma(b)$, thus we deduce $\sigma(ab) \xrightarrow{\star} \sigma(a'b)$.

   (b) The reduction takes place in $b$: symmetric to the previous case.

   (c) The reduction takes place at the root. It can be only a *Beta* reduction, and we must have $a = \lambda a'$ and $g = a'[b \cdot id]$. Then $\sigma(f) = (\lambda\sigma(a'))\sigma(b) \xrightarrow{Beta} \sigma(a')[\sigma(b) \cdot id]$. Now notice that $\sigma(a'[b \cdot id]) = \sigma(\sigma(a')[\sigma(b) \cdot id])$. Thus, by definition of $\beta$, we have: $\sigma(f) \xrightarrow{\beta} \sigma(g)$.

4. $f = a \cdot s$. We proceed similarly to the application case:

   (a) The reduction takes place in $a$ or $s$, say in $a$. Thus $g = a' \cdot s$ and $a \to a'$. Applying induction to $a$, we get $\sigma(a) \xrightarrow{\star} \sigma(a')$. We conclude similarly to the application case, noticing that $\sigma(a \cdot s) = \sigma(a) \cdot \sigma(s)$.

   (b) The reduction takes place at the root. It can only be a *VarShift* or a *SCons* redex:

   *VarShift* Then $f = 1 \cdot \uparrow$, $g = id$. The result follows obviously from $\sigma(f) = f$ and $\sigma(g) = g$.

   *SCons* Then $a = 1[s']$, $s = \uparrow \circ s'$ and $g = s'$. We have three cases:

      i. $\sigma(s')$ is not a *cons* and is different from $id$. Then $\sigma(f) = 1[\sigma(s')] \cdot (\uparrow \circ \sigma(s')) \xrightarrow{SCons} \sigma(g)$.

      ii. $\sigma(s') = id$. Then $\sigma(f) = 1 \cdot \uparrow \xrightarrow{VarShift} \sigma(g)$.

      iii. $\sigma(s') = a' \cdot s''$. Then $\sigma(f) = \sigma(g)$.

5. $f$ is a composition:

   (a) $f = \uparrow \circ s$. There are two cases:

      i. $s \to s'$ and $g = \uparrow \circ s'$. By induction, we have $\sigma(s) \xrightarrow{\star} \sigma(s')$. We distinguish three cases:

         A. $\sigma(s)$ is not a *cons* and is different from $id$. Then $\sigma(f) = \uparrow \circ \sigma(s) \xrightarrow{\star} \uparrow \circ \sigma(s') = \sigma(\uparrow \circ \sigma(s'))$ (notice the equality, which follows from Lemma 3.7) , which proves this case, since $\sigma(\uparrow \circ \sigma(s')) = \sigma(\uparrow \circ s') = \sigma(g)$.

B. $\sigma(s) = id$. Then $\sigma(s') = id$ follows from $\sigma(s) \xrightarrow{\star} \sigma(s')$. Hence $\sigma(f) = id = \sigma(g)$.

C. $\sigma(s) = a \cdot t$. There are three cases:

$\sigma(s') = a' \cdot t'$ and $t \xrightarrow{\star} t'$. Then we conclude $\sigma(f) \xrightarrow{\star} \sigma(g)$, since $\sigma(f) = t$ and $\sigma(g) = t'$.

The reduction $\sigma(s) \xrightarrow{\star} \sigma(s')$ reduces the root at some step, by *VarShift*, i.e. $a \xrightarrow{\star} 1$, $t \xrightarrow{\star} \uparrow$ and $\sigma(s') = id$. Then $\sigma(f) = t \xrightarrow{\star} \uparrow = \sigma(\uparrow \circ id) = \sigma(g)$.

The reduction $\sigma(s) \xrightarrow{\star} \sigma(s')$ reduces the root at some step, by *SCons*, i.e. $a \xrightarrow{\star} 1[t']$, $t \xrightarrow{\star} \uparrow \circ t'$ and $\sigma(s') = t'$. Then $\sigma(f) = t \xrightarrow{\star} \uparrow \circ t' = \sigma(\uparrow \circ t') = \sigma(g)$.

ii. $s = id$ and $g = \uparrow$: Then $\sigma(f) = \uparrow = \sigma(g)$.

(b) $f = \mathtt{x} \circ s$. The situation is similar to, but simpler than the previous case. We have either $s \to s'$, and then conclude from induction and $\sigma(\mathtt{x} \circ s) = \mathtt{x} \circ \sigma(s)$, or $s = id$ and $g = \mathtt{x}$, and then conclude from $\sigma(f) = f$ and $\sigma(g) = g$.

(c) $f = (s'' \circ s') \circ s$. We distinguish the following cases:

i. The reduction takes place in $s''$, $s'$ or $s$, say in $s''$. Thus $s'' \to t$ and $g = (t \circ s') \circ s$. We can apply induction to the $\sigma$ reduct $s'' \circ (s' \circ s)$ of $f$. Thus $\sigma(f) = \sigma(s'' \circ (s' \circ s)) \xrightarrow{\star} \sigma(t \circ (s' \circ s)) = \sigma(g)$.

ii. $s' = id$ and $g = s'' \circ s$. Then $f \xrightarrow{\star}_{\sigma} g$, hence $\sigma(f) = \sigma(g)$.

iii. $s = id$ and $g = s'' \circ s'$. We have $f \xrightarrow{\sigma} s'' \circ (s' \circ id)$ and $s'' \circ (s' \circ id) \xrightarrow{IdR} s'' \circ s'$. By induction, we get: $\sigma(f) = \sigma(s'' \circ (s' \circ id)) \xrightarrow{\star} \sigma(g)$.

(d) $f = (a \cdot t) \circ s$. We distinguish subcases:

i. The reduction takes place in $a$, $t$ or $s$. If it takes place in $a$ or $t$, we reason like in the corresponding subcase of $f = (s'' \circ s') \circ s$. If it takes place in $s$, then let $s \to s'$ and $g = (a \cdot t) \circ s'$. We have: $f \xrightarrow{\sigma} a[s] \cdot (t \circ s)$, and we can apply induction to $a[s]$ and $t \circ s$, and derive: $\sigma(a[s]) \xrightarrow{\star} \sigma(a[s'])$ and $\sigma(t \circ s) \xrightarrow{\star} \sigma(t \circ s')$. Combining, we get $\sigma(f) = \sigma(a[s]) \cdot \sigma(t \circ s) \xrightarrow{\star} \sigma(g)$.

ii. $s = id$ and $g = a \cdot t$. We have $f \xrightarrow{\sigma} a[id] \cdot (t \circ id)$, and by induction $\sigma(a[id]) \xrightarrow{\star} \sigma(a)$ and $\sigma(t \circ id) \xrightarrow{\star} \sigma(t)$, from which we get $\sigma(f) = \sigma(a[id]) \cdot \sigma(t \circ id) \xrightarrow{\star} \sigma(g)$.

iii. $a = 1$, $t = \uparrow$ and $g = s$. Then $f \stackrel{\sigma}{\to} 1[s] \cdot (\uparrow \circ s)$ and $1[s] \cdot (\uparrow \circ s) \stackrel{SCons}{\to} s$. By induction, we deduce: $\sigma(f) \stackrel{\star}{\to} \sigma(s) = \sigma(g)$.

iv. $a = 1[t']$, $s = \uparrow \circ t'$ and $g = t' \circ s$. We argue similarly to the previous subcase, noticing $f \stackrel{\star}{\to}_\sigma 1[t' \circ s] \cdot (\uparrow \circ (t' \circ s)) \stackrel{SCons}{\to} t' \circ s$.

6. $f$ is a closure. As for composition, we further decompose $f$:

   (a) $f = 1[s]$ or $f = \mathtt{X}[s]$. This situation is handled like $f = \uparrow \circ s$ or $f = \mathtt{x} \circ s$, respectively.

   (b) $f = (\lambda a)[s]$. There are two cases:

      i. The reduction occurs in $a$ or $s$. This case is handled like the corresponding subcase of $f = (s'' \circ s') \circ s$.

      ii. $s = id$ and $g = \lambda a$. We have: $f \stackrel{\star}{\to}_\sigma \lambda(a[1 \cdot \uparrow])$, and $\lambda(a[1 \cdot \uparrow]) \stackrel{VarShift}{\to} \lambda a$. Applying induction, we have: $\sigma(f) \stackrel{\star}{\to} \sigma(\lambda a)$.

   (c) $f = a[t][s]$. This case is handled like the case $f = (s'' \circ s') \circ s$.

   (d) $f = (ab)[s]$. There are three cases:

      i. The reduction occurs in $a$, $b$ or $s$, say in $s$: This case is handled as the corresponding subcase of $f = (a \cdot t) \circ s$.

      ii. $s = id$ and $g = ab$. This case is handled as the corresponding subcase of $f = (a \cdot t) \circ s$.

      iii. $a = \lambda a'$, $g = a'[b \cdot id][s]$. Notice that we have already finished the proof of the three last parts of the statement: we shall make use of this below. We have $f \stackrel{\star}{\to}_\sigma (\lambda(a'[1 \cdot (s \circ \uparrow)]))(b[s])$ and $(\lambda(a'[1 \cdot (s \circ \uparrow)]))(b[s]) \stackrel{Beta}{\to} a'[1 \cdot (s \circ \uparrow)][b[s] \cdot id]$. By induction, we have: $\sigma(f) \stackrel{\beta, IdR, Id^\star}{\to} \sigma(a'[1 \cdot (s \circ \uparrow)][b[s] \cdot id]) = \sigma(a'[b[s] \cdot (s \circ id)])$. We also have: $a'[b[s] \cdot (s \circ id)] \stackrel{IdR}{\to} a'[b[s] \cdot s]$. By the statement, part 2, we have: $\sigma(a'[b[s] \cdot (s \circ id)]) \stackrel{IdR, Id, VarShift^\star}{\to} \sigma(a'[b[s] \cdot s]) = \sigma(g)$. Concatenating the derivations, we get: $\sigma(f) \stackrel{\beta, IdR, Id^\star}{\to} \sigma(g)$.

This ends the proof. □

The following is the key lemma.

**Lemma 3.9** *The rules $\beta, Id, IdR, VarShift, SCons$ do not define a confluent system on $\sigma$-normal forms* [3].

[3] We are thankful to A. Rios for pointing out an error in a first version of this proof.

**Proof:** In the proof, $\rightarrow$ stands for a $\beta$, *Id, IdR, VarShift* or *SCons* step. Recall the term presented in [1] in support of the non confluence conjecture:

$$B = YC \text{ where } C = YV \text{ where } V = \lambda\lambda \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (\mathtt{1} \cdot id)] \cdot (\uparrow \circ (\mathtt{x} \circ ((\mathtt{21}) \cdot id)))]$$

where $Y = (\lambda\lambda \mathtt{1}(\mathtt{221}))(\lambda\lambda \mathtt{1}(\mathtt{221}))$ is Turing fixed point combinator. We shall actually need to consider a parameterized version of this term. For any substitution $s$, let:

$$B_s = YC_s \text{ where } C_s = YV_s \text{ where } V_s = \lambda\lambda \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (\mathtt{1} \cdot s)] \cdot (\uparrow \circ (\mathtt{x} \circ ((\mathtt{21}) \cdot s)))]$$

We shall omit subscripts $s$, when clear from the context. First observe, for any $a$, that:

$$(*) \quad Ca \xrightarrow{\star} VCa \xrightarrow{\star} \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (a \cdot s_1)] \cdot (\uparrow \circ (\mathtt{x} \circ ((Ca) \cdot s_1)))]$$

where $s_1 = \sigma(s \circ (a \cdot C \cdot id))$. Thus, for some $s'$:

$$\begin{array}{lll} B & \xrightarrow{\star} & CB \xrightarrow{\star} \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (B \cdot s')] \cdot (\uparrow \circ (\mathtt{x} \circ (CB \cdot s')))] \\ & \xrightarrow{\star} & \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (CB \cdot s')] \cdot (\uparrow \circ (\mathtt{x} \circ (CB \cdot s')))] \rightarrow \mathtt{X}[\mathtt{x} \circ (CB \cdot s')] = A \end{array}$$

and

$$B \xrightarrow{\star} CB \xrightarrow{\star} CA \text{ (since } B \xrightarrow{\star} A).$$

We show that $A$ and $C_sA$ cannot have a common reduct, whatever is $s$. A fortiori this will show that the reduction is not confluent. We proceed by contradiction, and suppose that a common reduct exists for some $s$. Let $K$ be a common reduct of minimum size (relatively to every $s$). We analyze all possible reductions of $CA$ to $K$. We shall denote by $\Box$ substitution occurrences which are irrelevant to the proof. It is not meant that they are all equal, they are just "black boxes".

First $C$ and $A$ are reduced independently. Let us examine the reductions from $C$:

$$\begin{array}{lll} C & \rightarrow & (\lambda\mathtt{1}(Y\mathtt{1}))V \text{ (no choice)} \\ & \xrightarrow{\star} & (\lambda\mathtt{1}Z)V \text{ (where } Y\mathtt{1} \xrightarrow{\star} Z) \\ & \xrightarrow{\star} & VC' \text{ (where } \sigma(Z[V \cdot id]) \xrightarrow{\star} C') \\ & \rightarrow & \lambda\mathtt{X}[\mathtt{1}[\mathtt{x} \circ (\mathtt{1} \cdot \Box)] \cdot (\uparrow \circ (\mathtt{x} \circ (\sigma(C'[\uparrow])\mathtt{1} \cdot \Box)))] \text{ (} Abs \text{ has been used)} \\ & \xrightarrow{\star} & \lambda\mathtt{X}[\mathtt{1}[\mathtt{x} \circ (\mathtt{1} \cdot \Box)] \cdot (\uparrow \circ (\mathtt{x} \circ (C'' \cdot \Box)))] \text{ (where } \sigma(C'[\uparrow])\mathtt{1} \xrightarrow{\star} C'') \end{array}$$

Notice that $\sigma(Z[V \cdot id]) \xrightarrow{\star} C'$ implies $C \xrightarrow{\star} \sigma(Z[V \cdot id])$, by proposition 3.8, since $C = \sigma((Y\mathtt{1})[V \cdot id])$. Hence $C \xrightarrow{\star} C'$.

Can surjective pairing be applied? Only if $\sigma(C'[\uparrow])\mathtt{1} \xrightarrow{\star} \mathtt{1}^4$, which implies $\sigma(C[\uparrow])\mathtt{1} \xrightarrow{\star} \mathtt{1}$, which in turn entails $\sigma(C[id])A = \sigma((C[\uparrow]\mathtt{1})[A \cdot id]) \xrightarrow{\star} A$. Notice that $\sigma(C_s[id]) = C_{s''}$ for some $s''$ (specifically, $s'' = \mathtt{1} \cdot ((\mathtt{1} \cdot (s \circ \uparrow)) \circ \uparrow)$). We have then:

$$\begin{array}{rcl} CA & \xrightarrow{\star} & (\lambda \mathtt{X}[\mathtt{x} \circ (\mathtt{1} \cdot \Box)])A' \ \text{(where } A \xrightarrow{\star} A') \\ & \rightarrow & \mathtt{X}[\mathtt{x} \circ (A' \cdot \Box)] \xrightarrow{\star} K \end{array}$$

The derivation from $\mathtt{X}[\mathtt{x} \circ (A' \cdot \Box)]$ to $K$ can only reduce $A'$ and $\Box$ independently: notice that the topmost *cons* node can never become the root of a *SCons* redex, since a reduct of $A'$ always begins with $\mathtt{X}$. Hence K has the form $\mathtt{X}[\mathtt{x} \circ (K' \cdot \Box)]$. But then $K'$ is a reduct of $A$, and a fortiori of $C_{s''}A$, which has a smaller size than $K$: contradiction.

Thus we may assume that the independent reduction of $C$ stops with $\lambda \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (\mathtt{1} \cdot \Box)] \cdot (\uparrow \circ (\mathtt{x} \circ (C'' \cdot \Box)))]$. We have:

$$\begin{array}{rcl} CA & \xrightarrow{\star} & \mathtt{X}[\mathtt{1}[\mathtt{x} \circ (A' \cdot \Box)] \cdot (\uparrow \circ (\mathtt{x} \circ (\sigma(C''[A' \cdot id]) \cdot \Box)))] \ \text{(where } A \xrightarrow{\star} A') \\ & \xrightarrow{\star} & \mathtt{X}[\mathtt{x} \circ (Q \cdot \Box)] \ \text{(where } A' \xrightarrow{\star} Q, \sigma(C''[A' \cdot id]) \xrightarrow{\star} Q) \\ & \xrightarrow{\star} & K \end{array}$$

This is the only way in which the reduction can proceed, as follows from the following observations.

- The only way to reach $K = \mathtt{X}[\mathtt{x} \circ \Box]$ is to apply surjective pairing at some stage in the substitution part of the top level closure.
- The form of $A'$, which begins with $\mathtt{X}$, prevents a reduct of $A' \cdot \Box$ to be a *SCons* redex.
- No reduct of $\sigma(C''[A' \cdot id]) \cdot \Box$ can be a *SCons* redex either. To show this, we observe that the reduct of a *SCons* redex which is in $\sigma$ normal form must have the form $\uparrow \circ \ldots \circ \uparrow \circ \mathtt{x} \circ \Box$. Hence if the root is rewritten using *SCons* during the reduction from $\sigma(C''[A' \cdot id]) \cdot \Box$, the end point of the derivation must have the form $\uparrow \circ \ldots \circ \uparrow \circ \mathtt{x} \circ \Box$, which cannot match a reduct of $(A' \cdot \Box)$.

[4] More slowly, surjective pairing can only be applied if $\sigma(C'[\uparrow])\mathtt{1} \cdot \Box$ and $\mathtt{1} \cdot \Box$ have a common reduct. On one hand, observe that the only possible reducts of $\mathtt{1} \cdot \Box$ are either $id$ (if *VarShift* is applied ) or again of the form $\mathtt{1} \cdot \Box$. On the other hand, to see that the only case to consider is $\sigma(C'[\uparrow])\mathtt{1} \xrightarrow{\star} \mathtt{1}$, observe that if $\sigma(C'[\uparrow])\mathtt{1} \cdot \Box$ reduces to a *SCons* redex, then $\sigma(C'[\uparrow])\mathtt{1} \xrightarrow{\star} \mathtt{1}[s]$ for some $s$ which reduces either to $id$ or to $\mathtt{1} \cdot \Box$. Since $\mathtt{1}[s]$ is a $\sigma$ normal form, $s$ can only be $\uparrow^n$ or $\mathtt{x} \circ \Box$ or $\uparrow^n \circ (\mathtt{x} \circ \Box)$, none of which can reduce to $id$ or to $\mathtt{1} \cdot \Box$.

Hence the derivation must reduce $A'$, $\sigma(C''[A' \cdot id])$ and the two "black box" substitutions independently. But observe that $Q$ is a common reduct of $A$ and $C_{s''}A$. We conclude as in the previous case by noticing that $Q \xrightarrow{\star} K'$ for some subterm $K'$ of $K$. We obtain again a contradiction to our size assumption, and this completes the proof. □

**Remark 3.1** *The counterexample used in proof of the last proposition suggested [14] a counterexample for λ-calculus extended with surjective pairing, leading to a simpler proof of non-confluence than the proofs known so far [29, 21]. The counterexample is:* $Y(Y(\lambda xy.D(D_1(Ey))(D_2(E(xy)))))$, *where* $D$ *is pairing and* $D_1$, $D_2$ *are the projections.*

We are now able to prove the main result of the subsection.

**Theorem 3.10** $\lambda\sigma_{SP}$ *is not confluent.*

**Proof:** By the interpretation lemma of section 1, and the previous lemmas. □

# 4 The confluent $\lambda\sigma$-calculus

In this section, we consider a confluent version of $\lambda\sigma$-calculus, initially in [23], called the $\lambda\sigma_{\Uparrow}$-calculus, or the confluent $\lambda\sigma$-calculus for simplicity. In addition to being confluent, this calculus presents the particularity that a rather simple proof of termination of the substitution rules could be found for it, in contrast to the situation with categorical combinators or $\sigma$.

We introduce the syntax and the rules (4.1) of $\lambda\sigma_{\Uparrow}$-calculus, we show the termination of the set of substitution rules (4.2), and the confluence of the full theory $\lambda\sigma_{\Uparrow}$ (4.3), by a technique which is different from the interpretation method. We end by showing that $\lambda\sigma_{\Uparrow}$-calculus relates well to the classical $\lambda$-calculus (4.4).

## 4.1 Syntax and rules

The main modification is to transform the rule:

$$(\lambda a)[s] \rightarrow \lambda(a[1 \cdot (s \circ \uparrow)])$$

into

$$(\lambda a)[s] \rightarrow \lambda(a[\Uparrow s])$$

in order to escape the non confluence of the $\lambda\sigma$-calculus. Terms and substitutions are now as follows:

| | |
|---|---|
| **Terms** | $a ::= \mathtt{X} \mid \mathtt{n} \mid ab \mid \lambda a \mid a[s]$ |
| **Substitutions** | $s ::= \mathtt{x} \mid id \mid \uparrow \mid a \cdot s \mid s \circ t \mid \Uparrow(s)$ |

The confluent $\lambda\sigma$-calculus $\lambda\sigma_{\Uparrow}$ is defined by the rules given in figure 5. $\sigma_{\Uparrow}$ is the rewriting system obtained by removing the rule (Beta) from the system $\lambda\sigma_{\Uparrow}$.

Remark that this system takes care of full De Bruijn's notation. Variables of the calculus are represented by any $\mathtt{n}$ instead of $\mathtt{1}[\uparrow^n]$. We could have followed the treatment that we adopted in previous sections. Nothing would have been changed for the results. Rules of the new calculus would have been less numerous, since we could have then erased rules *VarShift1*, *VarShift2*, *RVarCons*, *RVarLift1*, *RVarLift2*. Proofs are simply more difficult when taking $\mathtt{n}$ for any $n$.

The system is rather robust: rules for additional data types are possible. We take the instance of pairs, with a pairing operator (arity 2) and two projections *Fst* and *Snd* (arity 1) and we add the rules described in figure 6. This data type could have been tuples or lists instead. It can be proved that the confluence and termination properties still hold for this extension.

## 4.2 Substitution rules

We show that the sub-system $\sigma_{\Uparrow}$ is terminating and confluent. Remember that the termination of $\sigma$ was derived in [1] from the termination of the corresponding system for categorical combinators [22, 15]), which turned out to be a difficult problem. Surprisingly, the proof given here is quite simple. In fact, our calculus does not reproduce every possible calculations of $\sigma$ (but enough to simulate $\beta$-reductions, as we shall see). Moreover the rule *Lift2* for example does not correspond to a derivation in $\sigma$, but only to an equality.

**Proposition 4.1** *The system $\sigma_{\Uparrow}$ is locally confluent.*

**Proof:** First, notice that the rules preserve the sorts. Then, the untyped system is shown to be weakly confluent. For this, we use the KB system,

| | | | |
|---|---|---|---|
| *(Beta)* | $(\lambda a)b$ | $\rightarrow$ | $a[b \cdot id]$ |
| *(App)* | $(ab)[s]$ | $\rightarrow$ | $(a[s])\ (b[s])$ |
| *(Lambda)* | $(\lambda a)[s]$ | $\rightarrow$ | $\lambda(a[\Uparrow s])$ |
| *(Clos)* | $(a[s])[t]$ | $\rightarrow$ | $a[s \circ t]$ |
| *(VarShift1)* | $\mathtt{n}[\uparrow]$ | $\rightarrow$ | $\mathtt{n+1}$ |
| *(VarShift2)* | $\mathtt{n}[\uparrow \circ s]$ | $\rightarrow$ | $\mathtt{n+1}[s]$ |
| *(FVarCons)* | $\mathtt{1}[a \cdot s]$ | $\rightarrow$ | $a$ |
| *(FVarLift1)* | $\mathtt{1}[\Uparrow(s)]$ | $\rightarrow$ | $\mathtt{1}$ |
| *(FVarLift2)* | $\mathtt{1}[\Uparrow(s) \circ t]$ | $\rightarrow$ | $\mathtt{1}[t]$ |
| *(RVarCons)* | $\mathtt{n+1}[a \cdot s]$ | $\rightarrow$ | $\mathtt{n}[s]$ |
| *(RVarLift1)* | $\mathtt{n+1}[\Uparrow(s)]$ | $\rightarrow$ | $\mathtt{n}[s \circ \uparrow]$ |
| *(RVarLift2)* | $\mathtt{n+1}[\Uparrow(s) \circ t]$ | $\rightarrow$ | $\mathtt{n}[s \circ (\uparrow \circ t)]$ |
| *(AssEnv)* | $(s \circ t) \circ u$ | $\rightarrow$ | $s \circ (t \circ u)$ |
| *(MapEnv)* | $(a \cdot s) \circ t$ | $\rightarrow$ | $a[t] \cdot (s \circ t)$ |
| *(ShiftCons)* | $\uparrow \circ (a \cdot s)$ | $\rightarrow$ | $s$ |
| *(ShiftLift1)* | $\uparrow \circ \Uparrow(s)$ | $\rightarrow$ | $s \circ \uparrow$ |
| *(ShiftLift2)* | $\uparrow \circ (\Uparrow(s) \circ t)$ | $\rightarrow$ | $s \circ (\uparrow \circ t)$ |
| *(Lift1)* | $\Uparrow(s) \circ \Uparrow(t)$ | $\rightarrow$ | $\Uparrow(s \circ t)$ |
| *(Lift2)* | $\Uparrow(s) \circ (\Uparrow(t) \circ u)$ | $\rightarrow$ | $\Uparrow(s \circ t) \circ u$ |
| *(LiftEnv)* | $\Uparrow(s) \circ (a \cdot t)$ | $\rightarrow$ | $a \cdot (s \circ t)$ |
| *(IdL)* | $id \circ s$ | $\rightarrow$ | $s$ |
| *(IdR)* | $s \circ id$ | $\rightarrow$ | $s$ |
| *(LiftId)* | $\Uparrow(id)$ | $\rightarrow$ | $id$ |
| *(Id)* | $a[id]$ | $\rightarrow$ | $a$ |

Figure 5: The rewriting system $\lambda\sigma_{\Uparrow}$

| | | | |
|---|---|---|---|
| *(Fst)* | $Fst(<a,b>)$ | $\rightarrow$ | $a$ |
| *(Snd)* | $Snd(<a,b>)$ | $\rightarrow$ | $b$ |
| *(PairClos)* | $<a,b>[s]$ | $\rightarrow$ | $<a[s],b[s]>$ |
| *(FstClos)* | $Fst(a)[s]$ | $\rightarrow$ | $Fst(a[s])$ |
| *(SndClos)* | $Snd(a)[s]$ | $\rightarrow$ | $Snd(a[s])$ |

Figure 6: Rules for Pairing

developed at INRIA, which is an implementation of the Knuth-Bendix completion algorithm [31]. □

**Proposition 4.2** *The system $\sigma_{\Uparrow}$ is terminating.*

**Proof:** The termination of $\sigma$ is proved by a simple lexicographic ordering on two weights $P_1$ and $P_2$ defined on any terms or substitutions. These functions are defined by:

**Notation 4.1** *Let $P_1$ and $P_2$ be defined by:*

| | |
|---|---|
| $P_1(\mathtt{n}) = 2^n$ | $P_2(\mathtt{n}) = 1$ |
| $P_1(ab) = P_1(a) + P_1(b)$ | $P_2(ab) = P_2(a) + P_2(b) + 1$ |
| $P_1(\lambda a)) = P_1(a) + 2$ | $P_2(\lambda a) = 2 * P_2(a)$ |
| $P_1(a[s]) = P_1(a) * P_1(s)$ | $P_2(a[s]) = P_2(a) * (1 + P_2(s))$ |
| | |
| $P_1(a \cdot s) = P_1(a) + P_1(s)$ | $P_2(a \cdot s) = P_2(a) + P_2(s) + 1$ |
| $P_1(\uparrow) = 2$ | $P_2(\uparrow) = 1$ |
| $P_1(id) = 2$ | $P_2(id) = 1$ |
| $P_1(s \circ t) = P_1(s) * P_1(t)$ | $P_2(s \circ t) = P_2(s) * (1 + P_2(t))$ |
| $P_1(\Uparrow(s)) = P_1(s)$ | $P_2(\Uparrow(s)) = 4 * P_2(s)$ |
| | |
| $P_1(<a_1, a_2>) = P_1(a_1) + P_1(a_2)$ | $P_2(<a_1, a_2>) = P_2(a_1) + P_2(a_2) + 1$ |
| $P_1(Fst(a)) = P_1(a) + 2$ | $P_2(Fst(a)) = P_2(a) + 2$ |
| $P_1(Snd(a)) = P_1(a) + 2$ | $P_2(Snd(a)) = P_2(a) + 2$ |

□

**Lemma 4.3** *$P_1$ is decreasing on all the rules. Moreover it is strictly decreasing on (Lambda).*

**Proof:** The proof follows the usual technique using polynomial interpretations for proving termination of rewriting systems. First, it is easy to check that $P_1(a) \geq P_1(a')$ implies $P_1(C[a]) \geq P_1(C[a'])$ for any context $C[\,]$. Similarly for $s$. This is because polynomials are considered for values of variables greater than 1. Now, let us prove that the left hand sides of any rule of our system have a weight $L$ greater that the corresponding one $R$ of the right hand sides. We have to check every rule which is not a simplification rule and it will be simpler in this proof to identify terms $a$ and their weights $P_1(a)$.

*App* $L = as + bs = R$

*Lambda* $L = (a+2)s > as + 2 = R$ since $s \geq 2$ for every $s$.

*Clos* $L = ast = R$

*VarShift1* $L = 2^{n+1} = R$

*VarShift2* $L = 2^n(2s) = R$

*FVarCons* $L = 2(a+s) > a = R$

*FVarLift1* $L = 2s > 2 = R$

*FVarLift2* $L = 2st > 2t = R$

*RVarCons* $L = 2^{n+1}(a+s) > 2^n s = R$

*RVarLift1* $L = 2^{n+1}s = R$

*RVarLift2* $L = 2^{n+1}st = (2^n s)(2t) = R$

*AssEnv* $L = stu = R$

*MapEnv* $L = (a+s)t = (at) + (st) = R$

*ShiftCons* $L = 2(a+s) > s = R$

*ShiftLift1* $L = 2s = R$

*ShiftLift2* $L = 2st = R$

*Lift1* $\quad L = st = R$

*Lift2* $\quad L = stu = R$

*LiftEnv* $\quad L = s(a+t) > a + st = R$

*IdL* $\quad L = 2s > s = R$

*IdR* $\quad L = 2s > s = R$

*LiftId* $\quad L = 2 = R$

*Id* $\quad L = 2a > a = R$

*Fst* $\quad L = 2(a+b) > a = R$

*Snd* $\quad L = 2(a+b) > b = R$

*DPair* $\quad L = (a+b)s = (as) + (bs) = R$

□

**Lemma 4.4** *$P_2$ is strictly decreasing on all the rules but (Lambda), on which it is increasing.*

**Proof:** As previously, if $P_2(a) > P_2(a')$, then $P_2(C[a]) > P_2(C[a'])$ for every context $C[\,]$. Now, we use the same notations as for $P_1$. We work by cases on the rules:

*App* $\quad L = (a+b+1)(1+s) > a(1+s) + b(1+s) + 1 = R$
$\quad L - R = s$

*Lambda* $\quad L = (2a)(1+s) < 2a(1+4s) = R$
$\quad L - R = -6as$

*Clos* $\quad L = a(1+s)(1+t) > a(1+s(1+t))$
$\quad L - R = at$

*VarShift1* $\quad L = 1(1+1) = 2 > 1 = R$
$\quad L - R = 1$

*VarShift2* $\quad L = 1(1+1(1+s)) > 1(1+s) = R$
$\quad L - R = 1$

*FVarCons* $L = 1(1 + (a + s + 1)) > a = R$
$L - R = 2 + s$

*FVarLift1* $L = 1(1 + 4s) > 2 = R$
$L - R = 4s$

*FVarLift2* $L = 1(1 + 4s(1 + t)) > 1 + t = R$
$L - R = 4s + t(4s - 1)$

*RVarCons* $L = 1(1 + (a + s + 1)) > 1(1 + s) = R$
$L - R = 1 + a$

*RVarLift1* $L = 1(1 + 4s) > 1(1 + s(1 + 1)) = R$
$L - R = 2s$

*RVarLift2* $L = 1(1 + 4s(1 + t)) > 1(1 + s(1 + (1 + t))) = R$
$L - R = 2s + 3st$

*AssEnv* $L = s(1 + t)(1 + u) > s(1 + t(1 + u)) = R$
$L - R = st$

*MapEnv* $L = (a + s + 1)(1 + t) > a(1 + t) + s(1 + t) + 1 = R$
$L - R = t$

*ShiftCons* $L = 1(1 + (a + s + 1)) > s = R$
$L - R = 1 + a$

*ShiftLift1* $L = 1(1 + 4s) > s(1 + 1) = R$
$L - R = 2s$

*ShiftLift2* $L = 1(1 + 4s(1 + t)) > s(1 + 1(1 + t)) = R$
$L - R = 1 + 2s + 3st$

*Lift1* $L = 4s(1 + 4t) > 4s(1 + t) = R$
$L - R = 12st$

*Lift2* $L = 4s(1 + 4t(1 + u)) > 4s(1 + t)(1 + u) = R$
$L - R = 12st + 12stu - 4su$

*LiftEnv* $L = 4s(1 + (a + t + 1)) > a + s(1 + t) + 1 = R$
$L - R = a(4s - 1) + 7s - 1 + 3st$

*IdL* $L = 1(1 + s) > s = R$

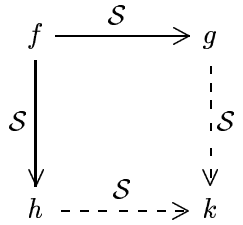
| | |
|---|---|
| *IdR* | $L = s(1+1) > s = R$ |
| *LiftId* | $L = 4 > 1 = R$ |
| *Id* | $L = a(1+1) > a = R$ |
| *Fst* | $L = 2(a+b+1) > a = R$ |
| *Snd* | $L = 2(a+b+1) > b = R$ |
| *Dpair* | $L = (a+b+1)(1+s) > 1 + a(1+s) + b(1+s) = R$<br>$L - R = s$ |

□

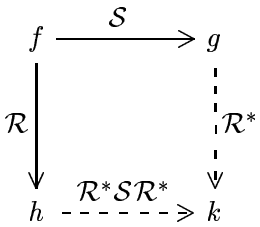
### 4.3 Confluence

Now we address the confluence of the full theory $\lambda\sigma_{\Uparrow}$. We shall use a general lemma coined in [42]:

**Lemma 4.5** *Let $\mathcal{R}$ and $\mathcal{S}$ be two relations defined on the same set $X$, $\mathcal{R}$ being confluent and strongly normalizing, and $\mathcal{S}$ verifying the diamond property, i.e. s.t. the following diagram holds, for any $f$,$g$,$h$ in $X$ (as usual, dashed arrows mean: "there exists $k$ s.t. ...):*

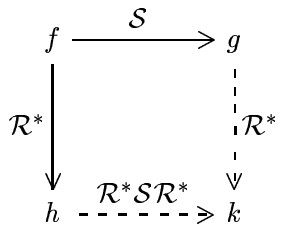



*Suppose moreover that the following diagram holds :*

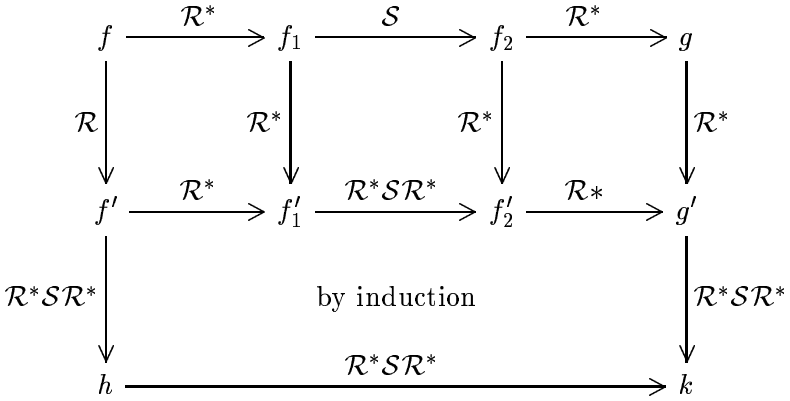



*Then the relation $\mathcal{R}^*\mathcal{S}\mathcal{R}^*$ is confluent.*

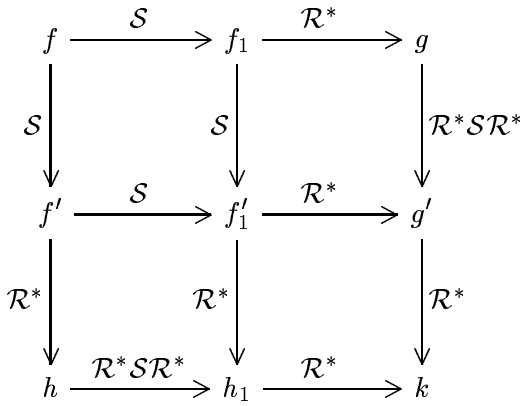
**Proof:** First, by induction on the $\mathcal{R}$-depth of $f$, we get




Then we conclude, again by induction on the $\mathcal{R}$-depth of $f$, and distinguishing between depth zero and non-zero.




and




□

We shall apply the lemma with the following datas. We take $\sigma_{\Uparrow}$ as $\mathcal{R}$ and $Beta\|$ as $\mathcal{S}$, where $Beta\|$ is the obvious parallelization of $Beta$ defined by:

$$\frac{}{a \to a} \qquad \frac{}{s \to s}$$

$$\frac{a \to a' \qquad b \to b'}{ab \to a'b'} \qquad \frac{a \to a' \qquad s \to s'}{a \cdot s \to a' \cdot s'}$$

$$\frac{a \to a'}{\lambda a \to \lambda a'} \qquad \frac{s \to s'}{\Uparrow(s) \to \Uparrow(s')}$$

$$\frac{a \to a' \qquad s \to s'}{a[s] \to a'[s']} \qquad \frac{s \to s' \qquad t \to t'}{s \circ t \to s' \circ t'}$$

$$\frac{a \to a' \qquad b \to b'}{(\lambda a)b \to a'[b' \cdot id]}$$

**Proposition 4.6** *$\sigma_{\Uparrow}$ and $Beta\|$ satisfy the conditions of lemma 4.5.*

**Proof:** The strong confluence of $Beta\|$ is obvious since $Beta$ by itself is a left linear system with no critical pairs [25, 26].

We check the second diagram. When the two steps from $f$ to $g$ and from $f$ to $h$ do not overlap (more technically do not make a critical pair), the lemma is easy since the system is left linear (i.e. without two occurences of a same variable on left hand sides of rules)[25, 26]. So we have to inspect every critical pair (in a sense slightly different from the standard one since $Beta\|$ is a parallel reduction). Since a strict subexpression of a $Beta\|$ redex can never overlap with a $\sigma_{\Uparrow}$ redex, it is sufficient to work by cases on the derivation from $f$ to $h$.

Case *App:* $f = (ab)[s] \xrightarrow{\sigma_{\Uparrow}} a[s]b[s] = h$. Then there are two cases:

1. $f = (ab)[s] \overset{Beta\|}{\to} (a'b')[s'] = g$ with $a \overset{Beta\|}{\to} a'$, $b \overset{Beta\|}{\to} b'$ and $s \overset{Beta\|}{\to} s'$. Then by definition of $Beta\|$, we have $a[s]b[s] \overset{Beta\|}{\to} a'[s']b'[s'] = k$. But also $g \xrightarrow{\sigma_{\Uparrow}} k$.

2. $f = ((\lambda a_1)b)[s] \overset{Beta\|}{\to} a_1'[b' \cdot id][s'] = g$ with $a_1 \overset{Beta\|}{\to} a_1'$, $b \overset{Beta\|}{\to} b'$ and $s \overset{Beta\|}{\to} s'$. Then $h = (\lambda a_1)[s]b[s]$. We must then take $h \xrightarrow{\sigma_{\Uparrow}} (\lambda a_1[\Uparrow s])b[s] = h_1$. Then

$h_1 \overset{Beta\|}{\rightarrow} a'_1[\Uparrow s'])[b'[s'] \cdot id] = h_2$. Then one checks easily that $h_2 \overset{\sigma^*_{\Uparrow}}{\longrightarrow} k$ and $g \overset{\sigma^*_{\Uparrow}}{\longrightarrow} k$ where $k = a'_1[b'[s'] \cdot s']$. This subcase is the only interesting one.

Case *Lambda:* $f = (\lambda a)[s] \overset{\sigma_{\Uparrow}}{\longrightarrow} \lambda(a[\Uparrow s]) = h$. Then $f \overset{Beta\|}{\rightarrow} g$ implies $g = (\lambda a')[s']$ with $a \overset{Beta\|}{\rightarrow} a'$, $s \overset{Beta\|}{\rightarrow} s'$. Thus $h \overset{Beta\|}{\rightarrow} \lambda(a'[\Uparrow s'])$ and $g \overset{\sigma_{\Uparrow}}{\longrightarrow} \lambda(a'[\Uparrow s'])$.

Case *Clos:* ... In fact all other cases are similar to the previous one. □

**Theorem 4.7** *The rewriting system $\lambda\sigma_{\Uparrow}$ is confluent.*

**Proof:** Notice that $\lambda\sigma_{\Uparrow} \subseteq \mathcal{R}^*\mathcal{S}\mathcal{R}^* \subseteq \lambda\sigma_{\Uparrow}{}^*$. □

## 4.4 Relation with classical $\lambda$-calculus

We pause to compare the technique used in this section to prove the confluence of $\lambda\sigma_{\Uparrow}$ to the interpretation method which we used to show the confluence of weak $\lambda\sigma$-calculus (or ground $\lambda\sigma$-calculus [1]). The order of the reductions in the conclusion of Lemma 2.5 is : $Beta$ followed by $\sigma$ normalization, followed by $Beta$... The order of the reductions indicated by the bottom line of the second picture of Lemma 4.5 is: some steps of $\sigma$, followed by some steps of $Beta$, followed by some steps of $\sigma$... The interpretation method relates better the classical calculus to the various explicit calculi. But, for the purpose of confluence, the method used in subsection 4.3 seems more direct: in the interpretation method, one has still to prove that the interpreted relation is confluent (cf. Lemma 2.4), while here some magic of diagrams is at work.

Since we want to relate the classical $\lambda$-calculus to the confluent $\lambda\sigma$-calculus, we shall also consider the interpretation method for the confluent $\lambda\sigma$-calculus, but we shall not give all the details.

Like in $\lambda\sigma$, we prove that it is possible to identify De Bruijn terms and ground terms in $\sigma_{\Uparrow}$-normal form, and that $\beta$-derivations correspond to one $Beta$ step followed by $\sigma_{\Uparrow}$-normalization. Moreover we prove that, for any derivation of a ground term $a$ to $b$, there is a corresponding $\beta$-derivation from $\sigma_{\Uparrow}(a)$ to $\sigma_{\Uparrow}(b)$.

First we describe the set of ground $\sigma_{\Uparrow}$-normal forms.

**Lemma 4.8** *Let $s$ be a ground substitution in $\sigma_{\Uparrow}$-normal form. Then, $s$ may have only one of the following forms:*

$s = id$,

$s = a \cdot t$ *where $a$, $t$ are $\sigma_{\Uparrow}$-normal,*

$s = \uparrow^n$, *that is* $\uparrow \circ (\ldots(\uparrow \circ \uparrow)\ldots)$,

$s = \Uparrow(t) \circ \uparrow^n$ *where $t$ is $\sigma_{\Uparrow}$-normal and $n \geq 0$.*

*Let $a$ be a ground term in $\sigma_{\Uparrow}$-normal form. Then, $a$ may have only one of the following forms:*

$a = \mathtt{n}$,

$a = bc$, *where $b$ and $c$ are $\sigma_{\Uparrow}$-normal,*

$a = \lambda(b)$ *where $b$ is $\sigma_{\Uparrow}$-normal,*

*Therefore, ground terms in $\sigma_{\Uparrow}$-normal form and $\lambda$-terms (in de Bruijn's notation) are the same.*

**Proof:**

1. The proof is by induction on the structure of the substitution $s$. We have only to examine the case $s = s_1 \circ s_2$. We have:
   (a) $s_1 \neq a \cdot s_{11}$ (otherwise $s$ would be a *MapEnv* redex).
   (b) $s_1 \neq s_{11} \circ s_{12}$ (otherwise $s$ would be a *AssEnv* redex).
   (c) if $s_1 = \uparrow^n$, then $n$ must be equal to 1 and $s_2$, which is in $\sigma_{\Uparrow}(\Lambda\sigma^o_{\Uparrow})$ by induction, can only be $\uparrow^p$ (otherwise $s$ would be a *ShiftCons* or *ShiftLift1* or *ShiftLift2* redex).
   (d) if $s_1 = \Uparrow(s_{11})$, then $s_2$ can only be $\uparrow^n$ (otherwise $s$ would be a *Lift1* or *Lift2* redex).
2. We prove, by induction on the structure, that a term $a$ in $\sigma_{\Uparrow}(\Lambda\sigma^o_{\Uparrow})$ cannot contain any substitution.
   (a) $a = n[s]$ where $s \in \sigma_{\Uparrow}(\Lambda\sigma^o_{\Uparrow})$. This case cannot appear: whatever $s$ is, by part 1, $n[s]$ contains a redex.
   (b) $a = a_1 a_2$ or $a = \lambda(a_1)$. Just use induction.
   (c) $a = b[s]$. This case cannot appear as $b[s]$ should contain no *Lambda* or *App* or *Closure* redex.

□

Let $\beta_{\Uparrow}$ be defined on $\lambda$-terms by: $a \xrightarrow{\beta_{\Uparrow}} b$ iff $a \xrightarrow{Beta} c$ and $b = \sigma_{\Uparrow}(c)$.

We prove the counterpart of the first statement of proposition 3.3, namely that $\beta$ and $\beta_{\Uparrow}$ may also be identified. It is convenient to use yet another definition of $\beta$-reduction (in abstract notation) which was proposed by De Bruijn, and used later on in works on categorical combinators.

**Definition 4.9** The substitution $a\{\mathtt{1} \leftarrow b\}$ is defined inductively by:

$$
\begin{array}{lll}
(ac)\{\mathtt{n} \leftarrow b\} & = & a\{\mathtt{n} \leftarrow b\}\ c\{\mathtt{n} \leftarrow b\} \\
\lambda a\{\mathtt{n} \leftarrow b\} & = & \lambda(a\{\mathtt{n+1} \leftarrow b\}) \\
\mathtt{m}\{\mathtt{n} \leftarrow b\} & = & \mathtt{m} - \mathtt{1} \quad \text{if} \quad \mathtt{m} > \mathtt{n} \\
 & & t_0^n(b) \quad \text{if} \quad \mathtt{m} = \mathtt{n} \\
 & & \mathtt{m} \quad \text{if} \quad \mathtt{m} < \mathtt{n}
\end{array}
$$

where $t_i^n(b)$ is defined by:

$$
\begin{array}{lll}
t_i^n(ac) = t_i^n(a)t_i^n(c) & & \\
t_i^n(\lambda a) = \lambda(t_{i+1}^n(a)) & & \\
t_i^n(\mathtt{m}) = & \mathtt{m} + \mathtt{n} - \mathtt{1} & \text{if} \quad \mathtt{m} \geq \mathtt{i} + \mathtt{1} \\
 & \mathtt{m} & \text{if} \quad \mathtt{m} \leq \mathtt{i}
\end{array}
$$

When performing the substitution, we count how many $\lambda$ nodes we cross between the occurrence of the redex and the occurrence currently reached: when getting $a\{\mathtt{n} \leftarrow b\}$, we crossed $(n-1)$ $\lambda$ nodes. Now in order to avoid capture, when meeting a De Bruijn number which has to be substituted, we have to update De Bruijn numbers of the substituted term. $t_0^n(b)$ says that $b$ will be placed under $n$ $\lambda$'s. $t_i^n(c)$ is obtained after crossing $i$ $\lambda$ between the root of the substituted term and its occurrence.

First we state that this notion of $\beta$-reduction is not exotic w.r.t. what we have developped so far (subsection 3.2).

**Proposition 4.10** *For any terms $\lambda$-terms $a$ and $b$, the two following relations are the same:*

1. *$\beta_\sigma$ (cf. Lemma 3.1),*
2. *$\beta_{DB}$, defined as the congruence generated by the axiom*

$$(\lambda a_1)a_2 \rightarrow a_1\{\mathtt{1} \leftarrow a_2\}\ .$$

**Proof:** Omitted. □

**Proposition 4.11** *For any $\lambda$-terms $a$ and $b$, $a \xrightarrow{\beta_{DB}} b$ iff $a \xrightarrow{\beta_{\Uparrow}} b$.*

**Proof:** Let $a$ be a term with a $\beta$-redex at occurrence $u$, that is: $a = C[u \longleftarrow (\lambda a_1)a_2]$. Suppose that:

$$a \xrightarrow{\beta} b \text{ and } a \xrightarrow{\beta_{\Uparrow}} c$$

Therefore

$$b = C[u \longleftarrow a_1\{\mathtt{1} \leftarrow a_2\}] \text{ and } c = \sigma_{\Uparrow}(C[u \longleftarrow a_1[a_2 \cdot id]]) \ .$$

Note that $c = C[u \longleftarrow \sigma_{\Uparrow}(a_1[a_2 \cdot id])]$ : since $a$ is a $\sigma_{\Uparrow}$-normal form, the replacement of the *Beta*-redex by the $\sigma_{\Uparrow}$-normal form of its reduct cannot create $\sigma_{\Uparrow}$-redexes at prefix occurrences. Therefore we only have to prove that $a_1\{\mathtt{1} \leftarrow a_2\} \equiv \sigma_{\Uparrow}(\ a_1[a_2 \cdot id]\ )$.
We prove the following equality for any terms $a$ and $b$ and for any $\mathtt{n}$:

$$a\{\mathtt{n} \leftarrow b\} = \sigma_{\Uparrow}(\ a\ [\ \Uparrow^{n-1}[b \cdot id]\ ]\ )$$

by induction on the structure of $\sigma_{\Uparrow}(a)$. The sole non trivial case is $a = \mathtt{m}$. The following equality, where $\mathtt{m} - \mathtt{p} \geq 1$, will help:

$$\mathtt{m}\ [\ \Uparrow^n(s)\ ] \xrightarrow{\sigma^*_{\Uparrow}} (\mathtt{m} - \mathtt{p})\ [\ \Uparrow^{n-p}(s) \circ \uparrow^p\ ]$$

Using this equality, we get:

$$\sigma_{\Uparrow}(\mathtt{m}\ [\ \Uparrow^{n-1}[b \cdot id]\ ]\ ) = \begin{array}{ll} (\mathtt{m} - \mathtt{1}) & \text{if } \ \mathtt{m} > \mathtt{n} \\ m & \text{if } \ \mathtt{m} < \mathtt{n} \\ \sigma_{\Uparrow}(\ b[\ \uparrow^{n-1}\ ]\ ) & \text{if } \ \mathtt{m} = \mathtt{n} \end{array}$$

It remains to prove for any term $b$:

$$\forall n \in b, t_i^n(b) = \sigma_{\Uparrow}(\ b\ [\ \Uparrow^i(\uparrow^{n-1})\ ]\ )$$

This is done by induction on $\sigma_{\Uparrow}(b)$, for all $n$. The only non trivial point is $b = \mathtt{m}$. Note that, if $\mathtt{m} - \mathtt{p} \geq 1$, we have: $\mathtt{m}[\ \Uparrow^i(\uparrow^n)\ ] \xrightarrow{\sigma^*_{\Uparrow}} (\mathtt{m} - \mathtt{p})[\ \Uparrow^{i-p}(\uparrow^n) \circ \uparrow^p\ ]$. With this equality we obtain:

$$\sigma_{\Uparrow}(\mathtt{m}\ [\ \Uparrow^i(\uparrow^n)\ ]\ ) = \begin{array}{ll} \mathtt{m} + \mathtt{n} & \text{if } \ \mathtt{m} \geq \mathtt{i} + \mathtt{1} \\ \mathtt{m} & \text{if } \ \mathtt{m} \leq \mathtt{i} \end{array}$$

□

Summarizing, we have considered four definitions of the $\beta$-reduction, among which three exactly equivalent ones are defined on De Bruijn terms.

These three relations are: $\beta_\sigma$ (Lemma 3.1), $\beta_{DB}$ (Proposition 4.10) and $\beta_{\Uparrow}$ . The equivalence with the classical $\beta$-reduction has been shown in Proposition 3.3. It is thus natural to call all these relations simply $\beta$.

The rest of the interpretation method can be carried in the same style as in section 2.3 (one is left to show that, for any ground $\lambda\sigma_{\Uparrow}$ term $a$, $a \stackrel{Beta}{\rightarrow} a'$ implies $\sigma_{\Uparrow}(a) \stackrel{\beta}{\rightarrow}^{\star} \sigma_{\Uparrow}(a')$).

# 5 Conclusion

We have extensively studied the confluence of the $\lambda\sigma$-calculus, and found a first theory ($\lambda\sigma_{\Uparrow}$ ) which is fully Church-Rosser. This point is interesting not only from a theoretical point of view, but also for an important practical aspect: it may be the right calculus for calculations on contexts of $\lambda$-expressions. It may be considered as an important extension of the classical $\lambda$-calculus. It seems also to be a good framework for the study of the abstract properties of implementations for functional languages, such as correctness or optimisation. But before making a full use of the confluent $\lambda\sigma$-calculus, a more extensive study of it should be done. In particular, it would be interesting to describe a calculus of shared terms in order to prove formally the soundness of "optimal" $\beta$ reducers [34, 19, 33, 28]. The theory of the $\lambda\sigma$-calculus could also be a good basis for the control of environments inside programming languages.

# References


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

[2] S. Abramsky, *The Lazy $\lambda$-calculus*, in Declarative Programming, D. Turner ed., 65-116, Addison Wesley, 1989.

[3] H. P. Barendregt, *The Lambda-Calculus*, vol 103, Elsevier Science Publishing Company, Amsterdam, 1984.

[4] H. P. Barendregt, *Pairing without conventional restraints*, Zeitschr. Math. Logik und Grundlagen der Math, 20, pp 289–306, 1974.

[5] G. Berry, J.-J. Lévy, *Minimal and Optimal Computations of Recursive Programs*, J.A.C.M. 26 (1), 1979.

[6] G. Boudol, *A Lambda-calculus for (Strict) Parallel Functions*, Information and Computation 108, 51-127, 1994.

[7] V. Breazu-Tannen, *A Combining Algebra and Higher-Order Types*, Proc. LICS 88, Edinburgh.

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

[9] N. de Bruijn, *Lambda-Calculus notation with namefree formulas involing symbols that represent reference transforming mappings* Indag. Math., 40, pp 348–356, 1978.

[10] A. Church, *The Calculi of Lambda-Conversion*, Ann. of Math. Studies, 6, 1941.

[11] G. Cousineau, P.-L. Curien, M. Mauny, *The Categorical Abstract Machine*, Science of Computer Programming 8, 173-202, 1987.

[12] P.-L. Curien, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Research Notes in Theoretical Computer Science, Pitman, London, 1986, Revised edition, Birkhaüser, 1993.

[13] P.-L. Curien, *An Abstract Framework for Environment Machines*, Theoret. Comput. Sci. 82, 389-402, 1991.

[14] P.-L. Curien, T. Hardin, *Yet Yet Another Counterexample for* $\lambda + SP$, Journal of Functional Programming 4(1), 113-115, 1994.

[15] P.-L. Curien, T. Hardin, A. Rios, *Strong Normalization of Substitutions*, in Proc. Mathematical Foundations of Computer Science, Prague, Lecture Notes in Computer Science 629, 1992.

[16] H. B. Curry, *Combinatory Logic*, vol 1, North-Holland, 1958.

[17] H. B. Curry, J. R. Hindley, and J. P. Seldin, *Combinatory Logic*, vol 2, North-Holland, 1972.

[18] N. Dershowitz, *Ordering for term rewriting system*, Theoret. Computer Sc., 17(3), pp 279-301, Mar 1982.

[19] J. Field, *On Laziness and Optimality in Lambda Interpreters*, ACM Conference on Principle of Programming Languages, San Francisco, 1990.

[20] T. Hardin, *Résultats de confluence pour les Règles fortes de la Logique Combinatoire Catégorique et Liens avec les Lambda-calculs*, thèse de Doctorat, Université de Paris 7, 1987.

[21] T. Hardin, *Confluence Results for the Pure Strong Categorical Logic CCL. λ-calculi as subsystems of CCL*, Theoret. Computer Sc., 65, pp 291–342, 1989.

[22] T. Hardin and A. Laville, *Proof of Termination of The Rewriting System Subst on C.C.L.* Theoret. Computer Sc., 46, pp 305–312, 1986.

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

[24] R. Hindley and J. Seldin, *Introduction to Combinators and λ-calculus*, Volume 1 of *London Mathematical Society Student texts*, Cambridge University Press, 1986.

[25] G. Huet, *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*, J.A.C.M., vol 27(4), pp 797–821, October 1980.

[26] G. Huet and J.-J. Lévy, "Computations in Orthogonal Rewriting Systems 1 et 2", Computational logic, Essays in Honor of Alan Robinson, ed. J.-L. Lassez & G. D. Plotkin, MIT Press, 1991.

[27] S. Kamin, J.-J. Lévy, *Two generalisations of recursive path orderings*, Unpublished note, 1980.

[28] V. Kathail, Optimal interpreters, PhD Thesis, MIT, 1990.

[29] J. W. Klop, *Combinatory Reduction Systems*, PhD, Mathematisch Centrum Amsterdam, 1982.

[30] J. W. Klop, R. de. Vrijer, *Unique Normal Forms for λ-calculus with Surjective Pairing*, Rapport 87-03, Centre of Mathematics and Computer Science, Amsterdam, 1987.

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

[32] J. Lambek, P.J. Scott, *Introduction to Higher Order Categorical Logic*, Cambridge University Press, Cambridge Amsterdam, 1987.

[33] J. Lamping, *An Algorithm for Optimal Lambda Calculus Reduction*, ACM Conference on Principle of Programming Languages, San Francisco, 1990.

[34] J.-J. Lévy, *Réductions correctes and optimales dans le Lambda- Calcul*, Thèse d'Etat, Université de Paris 7, 1978.

[35] L. Maranget, *Optimal Derivations in Weak Lambda-calculi and in Orthogonal Term Rewriting Systems*, POPL 91.

[36] M. Mauny, *Compilation des Langages Fonctionnels dans les Combinateurs Catégoriques*, Thèse de Troisième Cycle, Université Paris 7, Septembre 1985.

[37] C.-H. Luke Ong, *Fully Abstract Models of the Lazy $\lambda$-calculus*, Proc. FOCS 88.

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

[39] D. Turner, *A new implementation technique for applicative languages.* Software Practice and Experience, 9, pp 31–49, 1979.

[40] R. de Vrijer, *Surjective Pairing and Strong Normalisation: two themes in $\lambda$-calculus*, Dissertation, University of Amsterdam, 1987.

[41] P. Weis et al, *The CAML Reference Manual*, Projet FORMEL, INRIA-ENS, technical report, Version 2.6, 1989.

[42] H. Yokouchi, T. Hikita, *A rewriting system for categorical combinators with multiple arguments*, preprint, 1988.

[43] H. Yokouchi, *Relationship between $\lambda$-calculus and Rewriting Systems for Categorical Combinators*, Theoret. Computer Sc. 65, 1989.