A lambda-calculus with explicit weakening and explicit substitution

René David, Bruno Guillaume

Under consideration for publication in Math. Struct. in Comp. Science

A λ-calculus with explicit weakening and explicit substitution

RENÉ DAVID[1] and BRUNO GUILLAUME[1,2 †]

[1] *Laboratoire de Mathématiques*
Université de Savoie
F-73376 Le Bourget du Lac Cedex
[2] *Laboratoire de Recherche en Informatique*
Bât. 490 - Université Paris SUD
F-91405 Orsay Cedex

Received 3 April 2000

Since Melliès has shown that $\lambda\sigma$ (a calculus of explicit substitutions) does not preserve the strong normalization of the β-reduction, it became a challenge to find a calculus satisfying the following properties: step by step simulation of the β-reduction, confluence on terms with metavariables, strong normalization of the calculus of substitutions and preservation of the strong normalization of the λ-calculus. We present here such a calculus. The main novelty of the calculus (given with de Bruijn indices) is the use of *labels* that represent updating functions and correspond to explicit weakening. A typed version is also presented.

Contents

† This work was done at Université de Savoie and was finished in Université Paris SUD.

1. Introduction

Calculi of explicit substitutions are useful tools that fill the gap between the meta operation of substitution appearing in the β-reduction of the λ-calculus and its concrete implementation.

The most natural property such calculi have to satisfy is the simulation of the β-reduction (SIM): every β-reduction can be done in the new calculus and conversely this calculus does not introduce other reductions.

To have a good implementation of the λ-calculus, it is also natural to ask that no infinite reductions are created by the use of explicit substitutions. This is called the preservation of strong normalization (PSN). Melliès gave in (Melliès, 1995) a simply typed term with an infinite reduction in $\lambda\sigma$. This counter-example shows that $\lambda\sigma$ has not PSN.

Another important property is to have the confluence on terms with metavariables (MC): in proof assistants or theorem provers one has to consider proof trees with some unknown subtrees. To represent these proof trees, λ-terms with metavariables (corresponding to unknown parts of the tree) are necessary. The confluence on usual (closed) terms is easy to obtain but MC is much more difficult.

Since Melliès gave his counter-example, many calculi have been proposed but none of them satisfies simultaneously SIM, PSN and MC. Figure 1 gives some of them and their properties.

In order to satisfy both SIM and MC, rules for the interaction between substitutions are necessary. These rules are responsible for the lack of PSN in $\lambda\sigma$ and λs_e. In λd and $\lambda\sigma_n$, a weaker notion of composition is used and thus PSN is satisfied, but these rules are not strong enough to get MC.

The λs-calculus is the most natural calculus of explicit substitutions: it is the λ-calculus (with de Bruijn indices) where the substitution (σ^i) and the updating (ϕ^k_j) have been

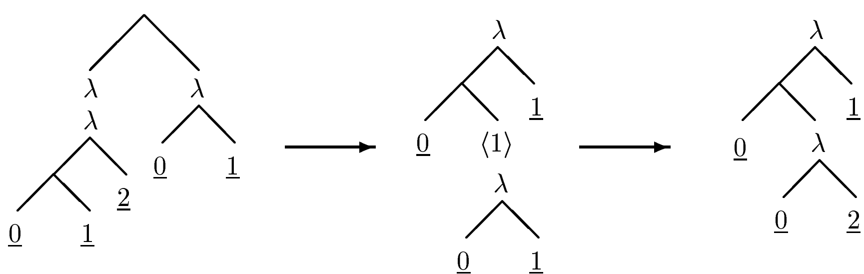
		SIM	PSN	MC
without interaction	$\lambda\nu$ (Benaissa et al., 1996)	Yes	Yes	No
	λs (Kamareddine and Ríos, 1995b)	Yes	Yes	No
	$\lambda\zeta$ (Muñoz, 1996; Muñoz, 1997)	Big step	Yes	Yes
with interaction	$\lambda\sigma$(Abadi et al., 1991)	Yes	No	Yes
	λs_e (Kamareddine and Ríos, 1997)	Yes	No	Yes
	λd (Ferreira et al., 1996)	Yes	Yes	No
	$SKInT$ (Goguen and Goubault-Larrecq, 1999)	Yes	Yes	Yes

Fig. 1. Calculi of explicit substitutions and their properties

internalized. The λs_e-calculus is obtained by adding new rules for the interaction of substitutions. This set of rules is the minimal one to get MC but unfortunately, λs_e does not satisfy PSN (Guillaume, 1999a).

In the following example, the β-reduction is done in two steps: first, the reduction of the β-redex and the propagation of the substitution and then, the propagation of the updating function. The $\langle 1 \rangle$ in the middle term means that the free indices in the term below must be increased by 1. This corresponds to the function ϕ_0^1 in λs_e.

Example 1.1.



The rules for the propagation of the updating functions are responsible for the lack of PSN in λs_e (Guillaume, 1999a). The key idea of our calculus is to keep the information about updating in terms rather than to move it down. In others words, we decide that (in the example above) the "right" reduct of the term is the second rather than the third one.

Recently, another solution which relies on a translation of λ-terms into sequent combinators has been proposed (Goguen and Goubault-Larrecq, 1999). Goguen and Goubault introduce a first order calculus (named $SKIn$) on the set of terms defined by:

$$t ::= x \mid I_m \mid K_m(t) \mid S_m(t,t)$$

where I_m, k_m and S_m are generalizations of the usual combinators I, K and S. The translation of the λ-term t in $SKIn$ is written t^* and the reverse one $[\![u]\!]$ for any $SKIn$-term u. They show that $t \longrightarrow_\beta u$ implies $t^* \longrightarrow^+_{SKIn} u^*$ but conversely, they only have that $t \longrightarrow_{SKIn} u$ implies $[\![t]\!] \longrightarrow^*_{\beta\eta} [\![u]\!]$. Unfortunately, with an example à la Melliès, they show that $SKIn$ is not strongly normalizing in the typed case and thus that it does not have PSN.

To recover the PSN, they define the $SKInT$-calculus on the same syntax but with less permissive rules. This second calculus has the expected properties (including PSN) but the relation with the λ-calculus is more complicated than for $SKIn$. The logic behind $SKInT$ is a fragment of the modal logic S4 called *near-intuitionistic logic*. The corresponding notion of "λ-calculus" is a closure calculus (named λ_{clos}) which is an extension of call-by-value (CBV) λ-calculus. The λ-calculus is translated in $SKInT$ in the following way: first, encode the λ-calculus in the CBV λ-calculus (using for example a continuation passing style (CPS) transformation), then use a translation from λ_{clos} to $SKInT$. Denoting by $L^*(t)$ the translation of the λ-term t in $SKInT$, they prove:

— if $t \longrightarrow_\beta u$ then $L^*(t) \longrightarrow^*_{SKInT} L^*(u)$;
— t and u are convertible if and only if $L^*(t)$ and $L^*(u)$ are convertible in $SKInT$.

The paper is organized as follow: we first introduce the λ_w-calculus (section 3)which is the usual λ-calculus (with de Bruijn indices) where terms may contain labels $\langle k \rangle$, then we give the λ_{ws}-calculus (section 4) which is obtained from the λ_w-calculus by making the substitutions explicit and by adding rules for interaction between substitutions.

The sections from 5 to 8 are devoted to the proofs of the main properties of the λ_{ws}-calculus. The most innovative section is the last one where the PSN is proved.

Warning: This paper is the complete version of the extended abstract presented in WESTAPP'99 (David and Guillaume, 1999). There, the λ_{ws}-calculus was called λ_l (l for label).

2. Preliminaries

We give here some definitions and useful lemmas about rewriting systems. We also recall the rules for the usual β-reduction on λ-terms with de Bruijn indices and the explicit substitution calculus λs_e.

2.1. *Rewriting*

Definition 2.1 (Abstract rewriting systems). Let E be a set of terms and R be a set of rewriting rules. We denote by $\longrightarrow_R$ the binary relation on E defined by the contextual closure of the set of rules.

We also write $\longrightarrow^*_R$ (resp. $\longrightarrow^+_R$) for the transitive and reflexive closure, (resp. transitive closure) of $\longrightarrow_R$.

Definition 2.2 (Normal form). We say that $t \in E$ is an R-normal form if there are no terms u such that $t \longrightarrow_R u$. The set of R-normal forms is denoted by $NF(R)$.

Definition 2.3 (Normalization).

— A term $t \in E$ is strongly normalizable if there is no infinite R-reduction of t, i.e. if every sequence $t \longrightarrow_R t_1 \longrightarrow_R t_2 \ldots$ is finite. The set of R-strongly normalizable terms is denoted by $SN(R)$. If $SN(R) = E$, we say that the reduction R is strongly normalizing.

— A term t is weakly normalizable if there is a finite reduction $t \longrightarrow_R^* u$ where u is an R-normal form. The set of R-weakly normalizable terms is denoted by $WN(R)$. If $WN(R) = E$, we say that the reduction R is weakly normalizing.

Definition 2.4 (Confluence).

— A reduction $\longrightarrow_R$ is confluent if, for $t, u, v \in E$ such that $t \longrightarrow_R^* u$ and $t \longrightarrow_R^* v$ there is w such that $u \longrightarrow_R^* w$ and $v \longrightarrow_R^* w$.
— A reduction $\longrightarrow_R$ is locally confluent if, for $t, u, v \in E$ such that $t \longrightarrow_R u$ and $t \longrightarrow_R v$ there is w such that $u \longrightarrow_R^* w$ and $v \longrightarrow_R^* w$.
— A reduction $\longrightarrow_R$ is strongly confluent if, for $t, u, v \in E$ such that $t \longrightarrow_R u$ and $t \longrightarrow_R v$ there is w such that $u \longrightarrow_R w$ and $v \longrightarrow_R w$.

Remark 2.5. The reduction $\longrightarrow_R$ is confluent if and only if the reduction $\longrightarrow_R^*$ is strongly confluent.

Lemma 2.6 (Newman's lemma). If the reduction $\longrightarrow_R$ is strongly normalizable and locally confluent, then it is confluent.

The following lemmas will be used in section 8. The second one is a particular case of the first one.

Lemma 2.7 (Projection lemma). Let R, S be reductions on E and F respectively and $\succcurlyeq$ be a binary relation on $E \times F$. Assume that:

— $R = R_1 \cup R_2$.
— R_1 is strongly normalizing.
— If $t \longrightarrow_{R_1} t'$ and $t \succcurlyeq u$ then there is u' such that $u \longrightarrow_S^* u'$ and $t' \succcurlyeq u'$.
— If $t \longrightarrow_{R_2} t'$ and $t \succcurlyeq u$ then there is u' such that $u \longrightarrow_S^+ u'$ and $t' \succcurlyeq u'$.

Let $t \in E$, $u \in F$ with $t \succcurlyeq u$. If $u \in SN(S)$ then $t \in SN(R)$.

Proof. From an infinite R-reduction of t, we can construct an infinite S-reduction of u:

$$\begin{array}{ccccccccc}
t = t_0 & \xrightarrow[R_1]{*} & t'_0 & \xrightarrow[R_2]{} & t_1 & \xrightarrow[R_1]{*} & t'_1 & \longrightarrow & \cdots \\
\curlyvee & & \curlyvee & & \curlyvee & & \curlyvee & & \\
u = u_0 & \xrightarrow[S]{*} & u'_0 & \xrightarrow[S]{+} & u_1 & \xrightarrow[S]{*} & u'_1 & \longrightarrow & \cdots
\end{array}$$

□

The next lemma corresponds to the particular case where R contains the equality (i.e. for all t, we have $t \succcurlyeq t$) and $S = R_2$.

Lemma 2.8 (Simulation lemma). Let $R = R_1 \cup R_2$ be a reduction on the set E and $\succcurlyeq$ be a binary relation on $E \times E$. Assume that:

— For all $t \in E$, we have $t \succcurlyeq t$.
— R_1 is strongly normalizable.
— If $t \longrightarrow_{R_1} t'$ and $t \succcurlyeq u$ then there is u' such that $u \longrightarrow_{R_2}^* u'$ and $t' \succcurlyeq u'$.
— If $t \longrightarrow_{R_2} t'$ and $t \succcurlyeq u$ then there is u' such that $u \longrightarrow_{R_2}^+ u'$ and $t' \succcurlyeq u'$.

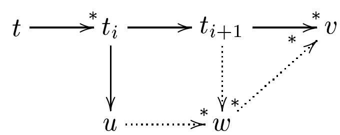
Then $SN(R) = SN(R_2)$.

The lemma 2.10 is an adaptation of a result given in (Klop, 1992). The original result is that a rewriting system which is locally confluent, weakly normalizing and increasing (there is a measure which is strictly increased by reduction) is also strongly normalizing. In lemma 2.10, the measure is only increasing (not strictly) but we have the additional hypothesis that reductions which leave the measure unchanged are strongly normalizing.

Lemma 2.9. Let R be a locally confluent reduction, t be a normalizable term and v be a normal form of t. Assume $t \notin SN(R)$. Then, there is a term $u \notin SN(R)$ such that $t \longrightarrow_R^+ u$ and v is a normal form of u.

Proof. Let $t = t_0 \longrightarrow t_1 \longrightarrow \dots \longrightarrow t_n = v$ a derivation from t to v. Let i be such that $t_i \notin SN(R)$ and $t_{i+1} \in SN(R)$ and u be a term such that $t_i \longrightarrow u$ and $u \notin SN(R)$.

Since R is locally confluent there is a term w such that $u \longrightarrow^* w$ and $t_{i+1} \longrightarrow^* w$. Since $t_{i+1} \in SN(R)$ and R is locally confluent, t_{i+1} has a unique R-normal form v and thus v also is a normal form of w. Finally, we have $t \longrightarrow^+ u$ and v is an R-normal form of u.



□

Lemma 2.10 (Increasing reductions). Let $R = R_1 \cup R_2$ and $|\cdot|$ be a measure such that:

— R_1 is strongly normalizing.
— If $t \longrightarrow_{R_1} t'$ then $|t| = |t'|$.
— If $t \longrightarrow_{R_2} t'$ then $|t| < |t'|$.
— R is weakly normalizing.
— R is locally confluent.

Then R is strongly normalizing.

Proof. Assume there is a term t which is not R-strongly normalizable. The weak normalization of the R-reduction ensures that t has an R-normal form v. By lemma 2.9 we can construct an infinite derivation: $t = t_0 \longrightarrow^+ t_1 \longrightarrow^+ \dots \longrightarrow^+ t_i \longrightarrow^+ \dots$ such that v is an R-normal form of each t_i. Since R_1 is strongly normalizing, there are infinitely many R_2-reductions in this derivation. Thus, $|t_j| > |v|$ for some j. This gives a contradiction since $t_j \longrightarrow_R^* v$ and thus $|t_j| \leq |v|$. □

2.2. *The λ-calculus with de Bruijn indices: the λ_{db}-calculus*

We will use de Bruijn representation of λ-terms where the first index is $\underline{0}$ and not $\underline{1}$. This will simplify notations in the next sections and, this is more natural with respect to the typed calculus. For instance, the λ-term $\lambda x \lambda y(x\ y)$ is written $\lambda\lambda(\underline{1}\ \underline{0})$.

Substitutions will be written on the left of the terms (for example $\{x := u\}t$ means t where x is substituted by u): this corresponds to the tree representation of terms and we believe this is easier to read.

Terms of the λ_{db}-calculus are defined by:

$$t ::= \underline{n} \mid \lambda t \mid (t\ t) \quad \text{with} \quad n \in \mathbb{N}$$

The β-reduction is given by the next definition. $\{i := u\}$ (the substitution) and ϕ (the updating function) are meta functions, i.e. are not in the syntax of the calculus.

Definition 2.11. The λ_{db}-calculus is defined by the rule:

$$(\lambda t\ u) \longrightarrow_{\lambda_{db}} \{0 := u\}t$$

with:

$$\{i := u\}\lambda t = \lambda\{i+1 := u\}t \qquad \phi_i^j(\lambda t) = \lambda \phi_{i+1}^j(t)$$

$$\{i := u\}(t\ v) = (\{i := u\}t\ \{i := u\}v) \qquad \phi_i^j(t\ u) = (\phi_i^j(t)\ \phi_i^j(u))$$

$$\{i := u\}\underline{n} = \begin{cases} \underline{n} & \text{if} \quad n < i \\ \phi_0^i(u) & \text{if} \quad n = i \\ \underline{n-1} & \text{if} \quad n > i \end{cases} \qquad \phi_i^j(\underline{n}) = \begin{cases} \underline{n} & \text{if} \quad n < i \\ \underline{n+j} & \text{if} \quad n \geq i \end{cases}$$

It is well known that this reduction is isomorphic to the usual β-reduction on λ-terms modulo α-equivalence (Kamareddine and Ríos, 1998).

2.3. *The λs-calculus and the λs_e-calculus*

The λs-calculus and the λs_e-calculus were introduced and studied by Kamareddine and Ros (Kamareddine and Ríos, 1995a; Kamareddine and Ríos, 1997). They both use the same syntax. The λs-calculus is obtained naturally from the λ_{db}-calculus by writing explicitly the substitutions and the updating functions.

$$t ::= \underline{n} \mid \lambda t \mid (t\ t) \mid [i := t]t \mid \langle i, j\rangle t \quad \text{with} \quad n, i, j \in \mathbb{N}$$

Remark 2.12. In the papers by Kamareddine and Ros, the first De Bruijn index is $\underline{1}$ whereas we use $\underline{0}$. The term $[i := u]t$ correspond to the term $t\sigma^{i+1}u$ in the original syntax and $\langle i, j\rangle t$ correspond to $\varphi_{i+1}^j(t)$.

Rules are translation of the definition 2.11:

$$
\begin{array}{llcll}
(\beta) & (\lambda t\ u) & \longrightarrow & [0 := u]t & \\
(\sigma\lambda) & [i := u]\lambda t & \longrightarrow & \lambda[i+1 := u]t & \\
(\sigma a) & [i := u](t\ v) & \longrightarrow & ([i := u]t\ [i := u]v) & \\
(\sigma n_1) & [i := u]\underline{n} & \longrightarrow & \underline{n} & \text{if } n < i \\
(\sigma n_2) & [i := u]\underline{n} & \longrightarrow & \langle 0, i\rangle u & \text{if } n = i \\
(\sigma n_3) & [i := u]\underline{n} & \longrightarrow & \underline{n-1} & \text{if } n > i \\
(\varphi\lambda) & \langle i, j\rangle\lambda t & \longrightarrow & \lambda\langle i+1, j\rangle t & \\
(\varphi a) & \langle i, j\rangle(t\ u) & \longrightarrow & (\langle i, j\rangle t\ \langle i, j\rangle u) & \\
(\varphi n_1) & \langle i, j\rangle\underline{n} & \longrightarrow & \underline{n} & \text{if } n < i \\
(\varphi n_2) & \langle i, j\rangle\underline{n} & \longrightarrow & \underline{n+j} & \text{if } n \geq i
\end{array}
$$

This calculus lacks only the metaconfluence property. In order to recover this property, the reduction relation is extended to give the λs_e-calculus. The extra rules are:

$$
\begin{array}{llcll}
(\sigma\sigma) & [i := u][j := v]t & \longrightarrow & [j := [i-j := u]v][i+1 := u]t & \text{if } j \leq i \\
(\sigma\varphi_1) & [i := u]\langle j, k\rangle t & \longrightarrow & \langle j, k-1\rangle t & \text{if } j \leq i < j+k \\
(\sigma\varphi_2) & [i := u]\langle j, k\rangle t & \longrightarrow & \langle j, k\rangle[i-k := u]t & \text{if } j+k \leq i \\
(\varphi\sigma) & \langle j, k\rangle[i := u]t & \longrightarrow & [i := \langle j-i, k\rangle u]\langle j+1, k\rangle t & \text{if } i \leq j \\
(\varphi\varphi_1) & \langle i, j\rangle\langle k, l\rangle t & \longrightarrow & \langle k, j+l\rangle t & \text{if } k \leq i \leq k+l \\
(\varphi\varphi_2) & \langle i, j\rangle\langle k, l\rangle t & \longrightarrow & \langle k, l\rangle\langle i-l, j\rangle t & \text{if } k+l < i
\end{array}
$$

These extra rules are exactly the ones needed to get MC. The strong normalization of the substitution calculus (λs_e without the rule β) is an open question.

The PSN was conjectured but its failure has been shown in (Guillaume, 1999a). At first sight, the $\sigma\sigma$-rule seems to be the right rule to have PSN: everything is right with respect to the Melliès counter-example. The problem comes from the rules for the interaction between substitutions and updatings. The following example shows where the problem arises.

Example 2.13.

$$[4 := u][7 := v]\langle 3, 4\rangle t \longrightarrow_{\varphi\sigma_2,\sigma\varphi} [4 := u][3 := \langle 0, 4\rangle v]\langle 4, 4\rangle t$$

In the left-hand side, the substitution $[4 := u]$ should not interact with the substitution $[7 := v]$ (because $4 < 7$, the $\sigma\sigma$-rule does not apply). In the right-hand side, after two reduction steps, the two substitutions can now interact and produce a self-embedded term as in the Melliès counter-example. This phenomenon can be used to construct an infinite reduction of a simply typed λ-term. See (Guillaume, 1999a) for details.

3. The calculus with explicit weakening: λ_w

3.1. *Terms with labels*

We avoid the counter-example to the PSN property of the λs_e-calculus by adding to the usual syntax a new constructor that we call a *label* and which represents an updating information. The term t with label k (denoted by $\langle k\rangle t$) corresponds to the term t where all free indices have been increased by k (i.e. $\phi_0^k(t)$ in λs_e).

In the terms we are finally interested in, two successive labels are not allowed. We first define preterms without this restriction.

Definition 3.1. We define the set of λ_w-preterms by the following grammar:

$$t ::= \underline{n} \mid \lambda t \mid (t\ t) \mid \langle k\rangle t \quad \text{with} \quad n, k \in \mathbb{N}$$

The function E defined below gives the λ_{db}-representation of a λ-term represented by a preterm.

Definition 3.2. The function E is defined from the set of preterms to Λ_{db} by:

— $E(\underline{n}) = \underline{n}$
— $E(\lambda t) = \lambda E(t)$
— $E(t\ u) = (E(t)\ E(u))$
— $E(\langle k\rangle t) = \phi_0^k(E(t))$

where ϕ is the function from Λ_{db} to Λ_{db} defined by:

— $\phi_i^j(\lambda t) = \lambda \phi_{i+1}^j(t)$
— $\phi_i^j(t\ u) = (\phi_i^j(t)\ \phi_i^j(u))$
— $\phi_i^j(\underline{n}) = \begin{cases} \underline{n} & \text{if} \quad n < i \\ \underline{n+j} & \text{if} \quad n \geq i \end{cases}$

Definition 3.3. Λ_w is the set of terms given by the following grammar:

$$\begin{aligned} t &::= u \mid \langle k\rangle u \quad \text{with} \quad k \in \mathbb{N} \\ u &::= \underline{n} \mid \lambda t \mid (t\,t) \quad \text{with} \quad n \in \mathbb{N} \end{aligned}$$

It is easy to define a reduction to recover a λ_w-term from any λ_w-preterm. Let m be the reduction rule (called mixing):

$$\langle i\rangle\langle j\rangle t \longrightarrow \langle i+j\rangle t$$

This reduction is clearly confluent and strongly normalizable on the set of preterms. We denote by $m(t)$ the λ_w-term which is the m-normal form of the preterm t.

The following lemma ensures that the m-reduction does not change the meaning of terms.

Lemma 3.4. Let t, u be λ_w-preterms such that $t \longrightarrow_m u$, then $E(t) = E(u)$. In particular, for each preterm t, we have $E(t) = E(m(t))$.

Proof. By an easy induction on the construction of t. Use the fact that for any λ_{db}-term v, we have $\phi_0^k(\phi_0^l(v)) = \phi_0^{k+l}(v)$. □

3.2. *The* λ_w*-calculus*

Let $t = (\langle k\rangle\lambda u\ v)$. Since $E(t)$ is a redex, t must also be a redex. We thus need a rule to reduce a redex which contains a label and the substitution must record this label. The substitution $\{i/u, j\}$ means that the indices i must be replaced by $\langle i\rangle u$ and that there was a label $\langle j\rangle$ in the redex.

Note that, even if t and u are terms, $\{i/u, j\}t$ only is a preterm. This is why, in the next definition, the m-normal form has to be taken in the β-rules. In the final calculus, the m-rule will also be an explicit rule.

Definition 3.5. The λ_w-calculus is defined on the set Λ_w by the two rules:

$$\begin{array}{lrcl} (\beta_1) & (\lambda t u) & \longrightarrow & m(\{0/u, 0\}t) \\ (\beta_2) & (\langle k\rangle\lambda t u) & \longrightarrow & m(\{0/u, k\}t) \end{array}$$

with:

— $\{i/u, j\}\underline{n} = \begin{cases} \underline{n} & \text{if} \quad n < i \\ \langle i\rangle u & \text{if} \quad n = i \\ \underline{n + j - 1} & \text{if} \quad n > i \end{cases}$

— $\{i/u, j\}\lambda t = \lambda(\{i + 1/u, j\}t)$

— $\{i/u, j\}(t\ v) = ((\{i/u, j\}t)\ (\{i/u, j\}v))$

— $\{i/u, j\}\langle k\rangle t = \begin{cases} \langle k + j - 1\rangle t & \text{if} \quad i < k \\ \langle k\rangle(\{i - k/u, j\}t) & \text{if} \quad i \geq k \end{cases}$

3.3. *Simply typed* λ_w*-calculus*

As usual, types (denoted by $A, B, \ldots$) are constructed with basic types and $\to$. Contexts (denoted by $\Gamma, \Delta, \ldots$) are lists of types. $|\Gamma|$ denotes the length of Γ. The typing rules are given below (where $|\Gamma| = i$).

$$\frac{}{\Gamma, A, \Delta \vdash \underline{i} : A}\,(ax) \qquad \frac{A, \Gamma \vdash t : B}{\Gamma \vdash \lambda t : A \to B}\,(\to_i)$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\,(\to_e) \qquad \frac{\Delta \vdash t : A}{\Gamma, \Delta \vdash \langle i\rangle t : A}\,(weak)$$

The first three rules are the usual ones of the λ_{db}-calculus. The last rule introduces labels. A label corresponds to a weakening in the proof tree associated with the term. This is the motivation of the subscript *"w"* in the name of the calculus.

The proof of subject reduction is straightforward.

Theorem 3.6 (Subject reduction). Let $t, u \in \Lambda_w$. Assume $t \longrightarrow^*_{\lambda_w} u$ and $\Gamma \vdash t : A$. Then $\Gamma \vdash u : A$.

It is easy to check that if $\Gamma \vdash t : A$, then $\Gamma \vdash E(t) : A$. The following result follows then immediately from theorem 3.15 below.

Theorem 3.7 (Strong normalization). Every typed λ_w-terms is strongly normalizable.

3.4. λ_w *versus* λ_{db}

In this subsection we show that the λ_w-calculus corresponds to the usual notion of β-reduction. We need some easy lemmas. Their detailed proof can be found in (Guillaume, 1999b).

Remark 3.8. Let $t \in \Lambda_{db}$ and $i \in \mathbb{N}$. Then $\phi_i^0(t) = t$.

Lemma 3.9. Let $t \in \Lambda_{db}$. Then

1 If $k \leq i \leq k + l$ then $\phi_i^j(\phi_k^l(t)) = \phi_k^{j+l}(t)$.
2 If $i > k + l$ then $\phi_i^j(\phi_k^l(t)) = \phi_k^l(\phi_{i-l}^j(t))$.

Lemma 3.10. Let $t, u \in \Lambda_{db}$ and $i \leq k < i + j$. Then $\{k := u\}\phi_i^j(t) = \phi_i^{j-1}(t)$.

Lemma 3.11. Let $t, u \in \Lambda_{db}$.

1 If $i \geq k$ then $\phi_i^j(\{k := u\}t) = \{k := \phi_{i-k}^j(u)\}\phi_{i+1}^j(t)$.
2 If $i \leq k$ then $\phi_i^j(\{k := u\}t) = \{k + j := u\}\phi_i^j(t)$.

Lemma 3.12. Let $t, u \in \Lambda_{db}$ be such that $t \longrightarrow_{\lambda_{db}} u$. Then $\phi_i^j(t) \longrightarrow_{\lambda_{db}} \phi_i^j(u)$.

Lemma 3.13. Let $t, u' \in \Lambda_{db}$ be such that $\phi_i^j(t) \longrightarrow_{\lambda_{db}} u'$. Then there is a term $u \in \Lambda_{db}$ such that $t \longrightarrow_{\lambda_{db}} u$ and $\phi_i^j(u) = u'$.

3.4.1. *The λ_{db}-calculus simulates the λ_w-calculus*

The following lemma translates a λ_w-term with substitution into a λ_{db}-term with substitution.

Lemma 3.14. Let t, u be λ_w-preterms. Then $E(\{i/u, j\}t) = \{i := E(u)\}\phi_{i+1}^j(E(t))$.

Proof. By induction on t. If $t = \lambda v$ or $t = (v\ w)$, the result is trivial.

— If $t = \underline{n}$ and $n < i$ then $E(\{i/u, j\}\underline{n}) = \underline{n} = \{i := E(u)\}\phi_{i+1}^j(E(\underline{n}))$.

— If $t = \underline{i}$ then $E(\{i/u, j\}\underline{i}) = E(\langle i\rangle u) = \phi_0^i(E(u))$. We have also $\phi_{i+1}^j(E(\underline{i})) = \underline{i}$ and so $\{i := E(u)\}\phi_{i+1}^j(E(\underline{i})) = \phi_0^i(E(u))$.

— If $t = \underline{n}$ and $n > i$ then $E(\{i/u, j\}\underline{n}) = \underline{n + j - 1} = \{i := E(u)\}\phi_{i+1}^j(E(\underline{n}))$.

— If $t = \langle k\rangle v$ and $i \geq k$, then

$$
\begin{aligned}
E(\{i/u, j\}\langle k\rangle v) &= E(\langle k\rangle\{i - k/u, j\}v) \\
&= \phi_0^k(E(\{i - k/u, j\}v)) \\
&= \phi_0^k(\{i - k := E(u)\}\phi_{i-k+1}^j(E(v))) && \text{induction hypothesis} \\
&= \{i := E(u)\}\phi_0^k(\phi_{i-k+1}^j(E(v))) && \text{lemma 3.11(2)} \\
\{i := E(u)\}\phi_{i+1}^j(E(\langle k\rangle t)) &= \{i := E(u)\}\phi_{i+1}^j(\phi_0^k(E(v))) \\
&= \{i := E(u)\}\phi_0^k(\phi_{i-k+1}^j(E(v))) && \text{lemma 3.9(2)}
\end{aligned}
$$

— If $t = \langle k\rangle v$ and $i < k$, then

$$\begin{array}{rcll}\{i := E(u)\}\phi^{j}_{i+1}(E(\langle k\rangle v)) & = & \{i := E(u)\}\phi^{j}_{i+1}(\phi^{k}_{0}(E(v))) & \\ & = & \{i := E(u)\}\phi^{j+k}_{0}(E(v)) & \text{lemma 3.9(1)} \\ & = & \phi^{j+k-1}_{0}(E(v)) & \text{lemma 3.10}\end{array}$$

$$\begin{array}{rcl}E(\{i/u, j\}\langle k\rangle v) & = & E(\langle j+k-1\rangle v) \\ & = & \phi^{j+k-1}_{0}(E(v))\end{array}$$

□

The following result shows that the λ_w-reduction corresponds to the usual λ_{db}-reduction.

Theorem 3.15. Let $t, u \in \Lambda_w$. If $t \longrightarrow_{\lambda_w} u$, then $E(t) \longrightarrow_{\lambda_{db}} E(u)$.

Proof. By induction on t.

— If $t = \lambda v$ and $u = \lambda v'$, or $t = (v\ w)$ and $u = (v'\ w)$, or $t = (w\ v)$ and $u = (w\ v')$ with $v \longrightarrow_{\lambda_w} v'$, we use the induction hypothesis.

— If $t = \langle k\rangle v$ and $u = \langle k\rangle v'$ with $v \longrightarrow_{\lambda_w} v'$, then by induction hypothesis $E(v) \longrightarrow_{\lambda_{db}} E(v')$ and, using lemma 3.12, $E(t) = \phi^k_0(E(v)) \longrightarrow_{\lambda_{db}} \phi^k_0(E(v')) = E(u)$.

— If $t = (\lambda v\ w)$ and $u = m(\{0/w, 0\}v)$ then $E(t) = (\lambda E(v) E(w))$ and $E(t) \longrightarrow_{\lambda_{db}} \{0 := E(w)\}E(v)$.

$$\begin{array}{rcll}E(u) & = & E(m(\{0/w, 0\}v)) & \\ & = & E(\{0/w, 0\}v) & \text{lemma 3.4} \\ & = & \{0 := E(w)\}\phi^0_1(E(v)) & \text{lemma 3.14} \\ & = & \{0 := E(w)\}E(v) & \text{remark 3.8}\end{array}$$

Finally, $E(t) \longrightarrow_{\lambda_{db}} E(u)$.

— If $t = (\langle k\rangle\lambda v\ w)$ and $u = m(\{0/w, k\}v)$ then $E(t) = (\lambda\phi^k_1(E(v))E(w)) \longrightarrow_{\lambda_{db}} \{0 := E(w)\}\phi^k_1(E(v))$.

$$\begin{array}{rcll}E(u) & = & E(m(\{0/w, k\}v)) & \\ & = & E(\{0/w, k\}v) & \text{lemma 3.4} \\ & = & \{0 := E(w)\}\phi^k_1(E(v)) & \text{lemma 3.14}\end{array}$$

Finally, $E(t) \longrightarrow_{\lambda_{db}} E(u)$.

□

3.4.2. *The λ_w-calculus simulates the λ_{db}-calculus*

Conversely, we show that, if t is a λ_w-term such that $E(t)$ has a β-redex, then the reduction of this redex can always be simulated in λ_w.

Theorem 3.16. Let $t \in \Lambda_w$ and $u' \in \Lambda_{db}$ be such that $E(t) \longrightarrow_{\lambda_{db}} u'$. Then, there is a term $u \in \Lambda_w$ such that $t \longrightarrow_{\lambda_w} u$ and $E(u) = u'$.

Proof. By induction on t. The non trivial cases are the following:

— If $t = \langle k\rangle v$ then $E(t) = \phi^k_0(E(v))$. Since $E(t) \longrightarrow_{\lambda_{db}} u'$, lemma 3.13 gives a term w' such that $E(v) \longrightarrow_{\lambda_{db}} w'$ and $\phi^k_0(w') = u'$. By the induction hypothesis on v, we get a term w such that $v \longrightarrow_{\lambda_w} w$ and $E(w) = w'$. Let $u = \langle k\rangle w$, then $t \longrightarrow_{\lambda_w} u$ and $E(u) = \phi^k_0(E(w)) = \phi^k_0(w') = u'$.

— If $t = (\lambda v\, w)$ and $u' = \{0 := E(w)\}E(v)$ then let $u = \{0/w, 0\}v$, we get $t \longrightarrow_{\lambda_w} u$ and $E(u) = \{0 := E(w)\}\phi^0_1(E(v))$ (lemma 3.14), and finally $E(u) = u'$ (lemma 3.9(1)).

term	λ_{db}-calculus	λ_w-calculus
$(inf\ 16\ 20)$	1.441.824 steps 10.5 seconds	101.761 steps 0.9 seconds
$((30\ pred)\ 30)$	607.840 steps 5.6 seconds	38.420 steps 1.4 seconds
$(mult\ 100\ 200)$	142.026 steps 1.7 seconds	80.718 steps 1.6 seconds

Fig. 2. Comparison between the λ_{db}-calculus and the λ_w-calculus

— If $t = (\langle k\rangle\lambda v\ w)$ and $u' = \{0 := E(w)\}\phi_1^k(E(v))$ then let $u = \{0/w, k\}v$, we get $t \longrightarrow_{\lambda_w} u$ and $E(u) = \{0 := E(w)\}\phi_1^k(E(v))$ (lemma 3.14), and so $E(u) = u'$.

□

3.5. *Conclusion: λ_w versus λ_{db}*

In our final calculus (the λ_{ws}-calculus defined below), the normal forms of the calculus of substitution are terms of Λ_w and not the usual terms of λ_{db}. We actually think that this gives a better representation of λ-terms.

— The fact that, with labels, a λ-term is not uniquely represented is not a drawback since labels are intrinsic: a term can be put in any context (whatever its labeling is). Therefore in an implementation, the function E (cf. definition 3.2) would be useless. Moreover, if necessary, the algorithm to check whether two terms represent the same λ-term is clearly linear in the size of the terms. Also note that it does not cost more work to translate a labeled term into a term with variables than to translate a usual de Bruijn term.

— A label in a typed term corresponds to a weakening in the associated proof. In the normalization of a proof, it is natural to move cuts up to the axioms i.e. to propagate substitutions in terms but there is no reason to move weakenings up to the axioms i.e. to propagate labels in terms.

— We hope that labels will give more efficient implementations. Compared with implementations in the representation of de Bruijn, there are no steps of propagation of lifts and many steps of propagation of substitutions are avoided since substitutions are erased earlier when they are useless. A very small implementation of the de Bruijn calculus and the labeled calculus gives an idea of the difference between these two presentations.

Figure 2 gives the number of elementary reduction steps and the time of the reduction to normal form in both systems. These tests were made on a PC-133Mhz with Objective Caml. The integers are the Church numerals. The inf function of the first example is an efficient one given in (David, 1994).

b_1	$(\lambda t\, u)$	$\longrightarrow$	$[0/u, 0]t$	
b_2	$(\langle k \rangle \lambda t\, u)$	$\longrightarrow$	$[0/u, k]t$	
l	$[i/u, j]\lambda t$	$\longrightarrow$	$\lambda[i+1/u, j]t$	
a	$[i/u, j](t\, v)$	$\longrightarrow$	$(([i/u, j]t)\, ([i/u, j]v))$	
e_1	$[i/u, j]\langle k \rangle t$	$\longrightarrow$	$\langle j+k-1 \rangle t$	$i < k$
e_2	$[i/u, j]\langle k \rangle t$	$\longrightarrow$	$\langle k \rangle [i-k/u, j]t$	$k \leq i$
n_1	$[i/u, j]\underline{n}$	$\longrightarrow$	$\underline{n}$	$n < i$
n_2	$[i/u, j]\underline{n}$	$\longrightarrow$	$\langle i \rangle u$	$n = i$
n_3	$[i/u, j]\underline{n}$	$\longrightarrow$	$\underline{n+j-1}$	$i < n$
c_1	$[i/u, j][k/v, l]t$	$\longrightarrow$	$[k/[i-k/u, j]v, j+l-1]t$	$k \leq i < k+l$
c_2	$[i/u, j][k/v, l]t$	$\longrightarrow$	$[k/[i-k/u, j]v, l][i-l+1/u, j]t$	$k+l \leq i$
m	$\langle i \rangle \langle j \rangle t$	$\longrightarrow$	$\langle i+j \rangle t$	

Fig. 3. Rules of the λ_{ws}-calculus

4. The λ_{ws}-calculus

4.1. *Syntax and reduction rules for the λ_{ws}-calculus*

In this section, we give our new calculus. The syntax is obtained from the syntax of the λ_w-calculus (definition 3.5) by adding a constructor for substitutions. This definition is similar to the definition of the λs_e-calculus from the λ_{db}-calculus.

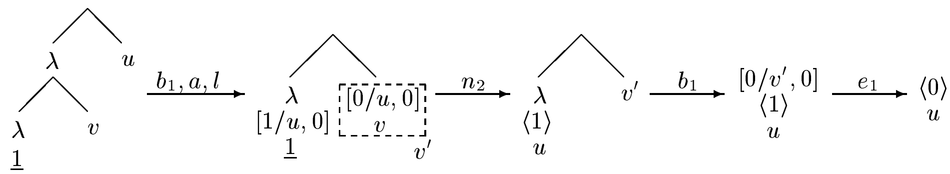
The set Λ_{ws} of terms of the λ_{ws}-calculus is defined by:

$$t ::= \underline{n} \mid \lambda t \mid (t\, t) \mid \langle k \rangle t \mid [i/t, j]t \quad \text{with} \quad n, i, j, k \in \mathbb{N}$$

Note that, as for the λ_w-calculus, two natural numbers are needed in each substitution: the second one keeps track of labels from redexes of the form $(\langle k \rangle \lambda t\, u)$. Also note that there is no restriction on nested labels: $\langle k \rangle \langle l \rangle t$ is a valid term of the λ_{ws}-calculus.

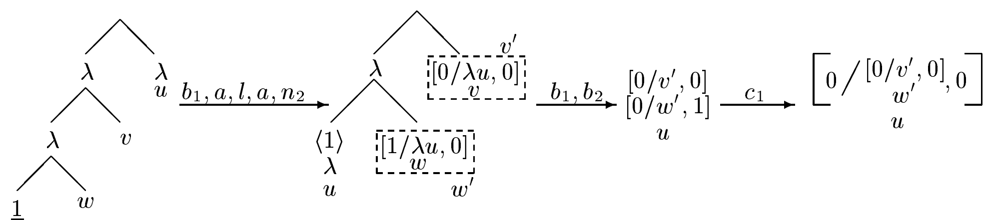
The set of rules is given in figure 3. The first two rules deal with β-redexes (with or without labels). The seven next rules come from the definition of the "implicit" substitution (definition 3.5). The composition rules c_1 and c_2 are needed for the confluence: they appear naturally to close the critical pairs a/b_1 and a/b_2 on the terms $[i/v, j](\lambda t\, u)$ and $[i/v, j](\langle k \rangle \lambda t\, u)$. Finally, the mixing rule m deals with nested labels. It has to be made explicit for the simulation of the β-reduction.

Example 4.1. The following example shows the use of the rule e_1. It erases a substitution when a label ensures that this substitution is useless in the term below.



In the last step, the substitution $[0/v',0]$ is erased in one step, independently of the complexity of u.

Example 4.2. The rule c_2 looks like the $\sigma\sigma$-rule of the λs_e-calculus. The rule c_1 is less common. This rule can be understood as the simultaneous use of c_2 and e_1:



In the last but one term, the substitution $[0/v',0]$ could be propagated in w' and u, but the index 1 in the second substitution ensures that $[0/v',0]$ is useless in u.

Notation 4.3. In the following, b will denote the reduction $b_1 \cup b_2$. In the same way, we define $e = e_1 \cup e_2$, $n = n_1 \cup n_2 \cup n_3$ and $c = c_1 \cup c_2$.

Definition 4.4. We define two sub-calculus on the set Λ_{ws} of terms:

- The ws-calculus is the λ_{ws}-calculus without the rules b_1 and b_2, i.e. the rules l, a, e, n, c and m.
- The p-calculus is the calculus of propagation of the substitutions i.e. the ws-calculus without the rule m, i.e. the rules l, a, e, n and c.

The ws-calculus allows the propagation of the substitutions and the contraction of successive labels. The p-calculus allows only the propagation of substitutions. The p-calculus is introduced for technical reasons: in the proof of PSN, working on p-normal forms rather than on ws-normal forms gives a shorter proof. The p-calculus is also used in the proof of the strong normalization of the ws-calculus.

Remark 4.5. For any $t \in \Lambda_{ws}$, we have $ws(t) = m(p(t))$, i.e. we can always postpone the mixing rule.

The complexity of a term is defined as usual as the number of constructors of the term:

Definition 4.6. The *complexity* of $t \in \Lambda_{ws}$ (denoted by $\text{cxty}(t)$) is defined by:

- $\text{cxty}(\underline{n}) = 1$
- $\text{cxty}(\lambda u) = 1 + \text{cxty}(u)$
- $\text{cxty}((u\ v)) = 1 + \text{cxty}(u) + \text{cxty}(v)$
- $\text{cxty}(\langle k \rangle u) = 1 + \text{cxty}(u)$

— $\mathrm{cxty}([i/u,j]v) = 1 + \mathrm{cxty}(u) + \mathrm{cxty}(v)$

4.2. *Typing rules for the λ_{ws}-calculus*

As usual, types (denoted by $A, B, \ldots$) are constructed with basic types and $\rightarrow$. Contexts (denoted by $\Gamma, \Delta, \ldots$) are lists of types. $|\Gamma|$ denotes the length of Γ.

The typing rules are the following (where $|\Gamma| = i$ and $|\Delta| = j$):

$$\frac{}{\Gamma, A, \Delta \vdash \underline{i} : A}\,(ax) \qquad \frac{A, \Gamma \vdash t : B}{\Gamma \vdash \lambda t : A \rightarrow B}\,(\rightarrow_i)$$

$$\frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\,(\rightarrow_e) \qquad \frac{\Delta \vdash t : A}{\Gamma, \Delta \vdash \langle i \rangle t : A}\,(weak)$$

$$\frac{\Delta, \Pi \vdash u : A \qquad \Gamma, A, \Pi \vdash t : B}{\Gamma, \Delta, \Pi \vdash [i/u, j]t : B}\,(cut)$$

We add the *cut* rule to the typing system of subsection 3.3. This rule is twofold : a (usual) cut and a weakening (Δ is added to the hypotheses for t). Intuitively, the context used to type $[i/u,j]t$ can be divided into three parts: the first one (Γ, of length i) is specific to t, the next one (Δ, of length j) is specific to u and the remaining one (Π) is common to t and u.

It is easy to check that the reduction rules of the λ_{ws}-calculus correspond naturally to the cut elimination process of the proof tree.

Theorem 4.7 (Subject reduction). Let $t, u \in \Lambda_{ws}$. If $t \longrightarrow_{\lambda_{ws}} u$ and $\Gamma \vdash t : A$, then $\Gamma \vdash u : A$.

Proof. By induction on t. We may assume that the reduction is at the root. Just check, for each rule, that the reduct can be typed, with the same type and the same hypothesis, as the redex. □

The rest of the paper is devoted to the untyped λs_e-calculus. We give here the normalization property of the typed calculus.

Theorem 4.8. Every typed λ_{ws}-term is weakly normalizable.

Proof. Let $t \in \Lambda_{ws}$ be typable. Theorem 4.7 ensures that $ws(t)$ (which exists by sections 5 and 6) is typable. The strong normalization of the typed λ-calculus and the preservation of the strong normalization of the λ_{ws}-calculus (section 8) ensure that $ws(t)$ is strongly normalizable for the λ_{ws}-calculus. Finally, t is weakly normalizable. □

It should be possible to prove the strong normalization of the typed calculus by the same kind of technique as in the proof of PSN. This result has been proved recently (Di Cosmo et al., 2000) by using a translation into proof nets, a technique introduced in (Di Cosmo and Kesner, 1997).

4.3. *Link with the λs_e-calculus*

Every λ_{ws}-term t can be translated in a λs_e-term (denoted by $t^\sharp$) in the following way. Note that there is no translation in the other way.

— $\underline{n}^\sharp = \underline{n}$
— $(\lambda t)^\sharp = \lambda t^\sharp$
— $(t\ u)^\sharp = (t^\sharp\ u^\sharp)$
— $(\langle k \rangle t)^\sharp = \langle 0, k \rangle t^\sharp$
— $[i/u, j]t = [i := u^\sharp]\langle i+1, j \rangle t^\sharp$

The λ_{ws}-calculus can be seen as a part of the λs_e-calculus where some reductions are forbidden. Intuitively, in λ_{ws}, an updating $\langle i, j \rangle$ may not move down, except if it appears at the root of the function part of a redex. In this case, the updating may cross the λ but the redex has to be contracted immediately after and this new updating must be linked to the substitution coming from the redex (i.e. they cannot move independently). The relation betwen both calculi is the following. If $t \longrightarrow_{\lambda_{ws}} u$ then $t^\sharp \longrightarrow^+_{\lambda s_e} u^\sharp$ and one step of λ_{ws}-reduction can be simulated by a fixed number (from 1 to 4, depending of the rule) of λs_e-reduction.

5. Strong normalization of the calculus of substitutions

In this section, we prove that the ws-calculus is strongly normalizing. This proof is inspired by the one Zantema gave for the strong normalization of the $\sigma\sigma$-rule of the λs_e-calculus (Zantema, 1998).

We first show, using the simulation lemma (lemma 2.8) that $SN(ws) = SN(p_2)$ where the p_2-calculus is the calculus defined by the set of rules that increase (not strictly) the complexity i.e. the rules l, a, e_2 and c (subsection 5.2). Then, using the increasing reductions lemma (lemma 2.10), we prove that the p_2-calculus is strongly normalizing (subsection 5.3). We finally get the theorem:

Theorem 5.1. The ws-calculus is strongly normalizing.

The complete proofs of the lemmas 5.6, 5.7 and of the propositions 5.9, 5.10 and 5.11 can be found in (Guillaume, 1999b).

5.1. *The substitutive contexts*

In the rest of this paper, the notion of "normal form" of a sequence of substitutions is useful. We call such a sequence *substitutive context*.

Notation 5.2. $\overline{\mathbb{N}}$ denotes the set $\mathbb{N} \cup \{-\infty, \infty\}$ with its natural ordering extended in such a way that $-\infty$ is the smallest element and ∞ is the greatest one. The addition is extended by $i + \infty = \infty$ and $i - \infty = -\infty$ for $i \in \mathbb{N}$ ($\infty - \infty$ is not defined).

We use the notation $*$ to represent a term about which nothing has to be known. The contexts have only one hole (denoted by $\{\!\!\{\cdot\}\!\!\}$). $C\{\!\!\{t\}\!\!\}$ denotes the context C in which the hole has been replaced by t.

Definition 5.3. A *substitutive context* is a context:
$S = [i_1/*, j_1] \dots [i_n/*, j_n]\{\!|\cdot|\!\}$ with $n \geq 0$ and $i_1 < \dots < i_n$.
We define:

— The *initial index* $i(S) \in \overline{\mathbb{N}}$ of S:

$$i(S) = \begin{cases} \infty & \text{if } n = 0 \\ i_1 & \text{if } n > 0 \end{cases}$$

— The *final index* $f(S) \in \overline{\mathbb{N}}$ of S:

$$f(S) = \begin{cases} -\infty & \text{if } n = 0 \\ i_n & \text{if } n > 0 \end{cases}$$

— The *height* $h(S) \in \mathbb{N}$ of S:

$h(S) = n$

— The *shift* $d(S) \in \mathbb{Z}$ of S:

$$d(S) = \left(\sum_{k=1}^{n} j_k\right) - n$$

It is important to note that (for technical reasons) we allow a substitutive context to be empty. When there is no ambiguity, we extend the usual notion of reduction on the terms to reduction on contexts.

Notation 5.4. If S is the substitutive context $[i_1/*, j_1] \dots [i_n/*, j_n]$ we will denote by:

— $[i/u, j]S$ the substitutive context $[i/u, j][i_1/*, j_1] \dots [i_n/*, j_n]\{\!|\cdot|\!\}$ if $i < i(S)$.
— $S[i/u, j]$ the substitutive context $[i_1/*, j_1] \dots [i_n/*, j_n][i/u, j]\{\!|\cdot|\!\}$ if $i > f(S)$.

Remark 5.5. Let S be a substitutive context such that $h(s) > 0$, then $f(S) \geq i(S)$. A trivial induction on $h(S)$ show that if $h(s) > 0$ then $i(S) \leq f(S) - h(S) + 1$.

The next two lemmas give the result of the "composition" of a substitution with a substitutive context. There are two cases: either the new substitution can "go through" the context (lemma 5.6), or the substitution is "integrated" in the context (lemma 5.7). These two cases are not disjoint: when the substitution goes through the context, we can choose to either integrate it at the end of the context (lemma 5.7) or keep it separated (lemma 5.6).

Lemma 5.6. Let S be a substitutive context, and $[i/u, j]$ be a substitution such that $i > d(S) + f(S)$. Then, there is a substitutive context S' such that: $[i/u, j]S \longrightarrow_c^* S'[i - d(S)/u, j]$, $d(S') = d(S)$ and $f(S') = f(S)$.

Proof. The proof is by induction on $h(S)$: we show that there is a substitutive context S' such that $[i/u, j]S \longrightarrow_c^* S'[i - d(S)/u, j]$, $d(S') = d(S)$, $f(S') = f(S)$ and $i(S') = i(S)$. □

Lemma 5.7. Let S be a substitutive context, and $[i/u, j]$ a substitution. Then there is a substitutive context S' such that $[i/u, j]S \longrightarrow_c^* S'$, $d(S') = d(S) + j - 1$ and

$$\begin{cases} f(S') = f(S) & \text{if } i \leq d(S) + f(S) \\ f(S') = i - d(S) & \text{if } i > d(S) + f(S) \end{cases}$$

Proof. The case $i > d(S)+f(S)$ is a reformulation of the previous lemma. Indeed, there is a substitutive context S'' such that $[i/u,j]S \longrightarrow_c^* S''[i-d(S)/u,j]$. Let $S' = S''[i-d(S)/u,j]$, we verify that S' is a substitutive context because $f(S'') = f(S) < i - d(S)$.

For the second point, remark that $i \leq d(S) + f(S)$ only if $h(S) > 0$. We then use an induction on $h(S)$ and show that there is a substitutive context S' such that $[i/u,j]S \longrightarrow_c^* S'$, $d(S') = d(S) + j - 1$, $f(S') = f(S)$ and $i(S') = \min(i, i(S))$.

□

The following lemma shows how we can erase the context when all the substitutions are useless. This also shows that $d(S)$ really is a shift.

Lemma 5.8. Let S be a substitutive context. If $f(s) < k$, then $S\{\!\!\{\langle k\rangle t\}\!\!\} \longrightarrow_{e_1}^* \langle k + d(S)\rangle t$.

Proof. By induction on $h(S)$. The case $h(S) = 0$ is trivial. If $h(S) > 0$, then $S = S'[i/u,j]$ with $i < k$ and $d(S) = d(S') + j - 1$ so

$$S\{\!\!\{\langle k\rangle t\}\!\!\} \longrightarrow_{e_1} S'\{\!\!\{\langle k + j - 1\rangle t\}\!\!\}$$

and we can use the induction hypothesis (because $f(S') < i \leq k-1$) and $f(S') < k+j-1$), we have

$$S'\{\!\!\{\langle k + j - 1\rangle t\}\!\!\} \longrightarrow_{e_1}^* \langle k + j - 1 + d(S')\rangle t = \langle k + d(S)\rangle t\,.$$

□

5.2. *Simulation of the ws-calculus in the* p_2*-calculus*

In order to prove that $SN(ws) = SN(p_2)$, we use intermediate reductions:

— The ws-calculus is defined by the rules l, a, e, n, c and m.
— The p-calculus (propagation) is defined by the rules l, a, e, n and c.
— The p_1-calculus is defined by the rules l, a, e_2, n and c.
— The p_2-calculus (rules which increase (not strictly) the complexity) is defined by the rules l, a, e_2 and c.

The proof is divided in three steps: $SN(ws) = SN(p)$, $SN(p) = SN(p_1)$ and $SN(p_1) = SN(p_2)$.

Proposition 5.9. $SN(ws) = SN(p)$.

Proof. $ws = p \cup m$. We use the simulation lemma (lemma 2.8) with $R_1 = m$, $R_2 = p$ and the relation $\succcurlyeq$ on $\Lambda_{ws} \times \Lambda_{ws}$ defined by:

— $\underline{k} \succcurlyeq \underline{k}$
— $\lambda t \succcurlyeq \lambda t'$ iff $t \succcurlyeq t'$
— $(tu) \succcurlyeq (t'u')$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$
— $\langle k\rangle t \succcurlyeq \langle k_1\rangle \ldots \langle k_n\rangle t'$ iff $t \succcurlyeq t'$, $n \geq 1$ and $k = k_1 + \ldots + k_n$
— $[i/u,j]t \succcurlyeq [i/u',j]t'$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$.

□

Proposition 5.10. $SN(p) = SN(p_1)$.

Proof. $p = p_1 \cup e_1$. We use lemma 2.8 with $R_1 = e_1$, $R_2 = p_1$, and the relation $\succcurlyeq$ defined by:

— $\underline{k} \succcurlyeq \underline{k}$
— $\lambda t \succcurlyeq \lambda t'$ iff $t \succcurlyeq t'$
— $(tu) \succcurlyeq (t'u')$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$
— $\langle k \rangle t \succcurlyeq S\{\!\!\{\langle k' \rangle t'\}\!\!\}$ with $t \succcurlyeq t'$ and S is a substitutive context such that $f(S) < k'$ and $k = k' + d(S)$
— $[i/u, j]t \succcurlyeq [i/u', j]t'$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$.

□

Proposition 5.11. $SN(p_1) = SN(p_2)$.

Proof. $p_1 = p_2 \cup n$. We use lemma 2.8 with $R_1 = n$, $R_2 = p_2$ and the relation $\succcurlyeq$ defined by:

— $\underline{k} \succcurlyeq t$ for all t
— $\lambda t \succcurlyeq \lambda t'$ iff $t \succcurlyeq t'$
— $(tu) \succcurlyeq (t'u')$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$
— $\langle k \rangle t \succcurlyeq v$ iff v is of one of the following form:
  – $v = \langle k \rangle t'$ and $t \succcurlyeq t'$
  – $v = S\{\!\!\{[k'/t', j]w\}\!\!\}$ with $f(S) < k'$, $k = k' + d(S)$ and $t \succcurlyeq t'$
— $[i/u, j]t \succcurlyeq [i/u', j]t'$ iff $t \succcurlyeq t'$ and $u \succcurlyeq u'$.

□

5.3. *Strong normalization of the p_2-calculus*

We prove the strong normalization of the p_2-calculus by using lemma 2.10 where the measure is the complexity and

— R_1 = the rules l, c_1 and e_2 (the complexity is left unchanged by these rules).
— R_2 = the rules a and c_2 (these rules increase the complexity).

We have to check:

— The reduction R_1 is strongly normalizable (proposition 5.12).
— The reduction p_2 is locally confluent (this is done by analyzing critical pairs).
— The reduction p_2 is weakly normalizing (proposition 5.15).

5.3.1. *Strong normalization of the rules that leave complexity unchanged*

We use a measure which decreases by R_1-reduction. This measure is the sum of the complexities of the subterms which are below each substitution of the term.

Proposition 5.12. The R_1-reduction is strongly normalizing.

Proof. The measure $||\cdot||$ defined below strictly decreases by l, c_1 and e_2-reduction:

— $||\underline{n}|| = 0$
— $||\lambda t|| = ||t||$

— $||(t\,u)|| = ||t|| + ||u||$
— $||\langle k\rangle t|| = ||t||$
— $||[i/u,j]t|| = ||t|| + ||u|| + \text{cxty}(t)$.

□

5.3.2. *Weak normalization of the p_2-calculus*

Definition 5.13. We define a binary relation $\Uparrow$ on $\Lambda_{ws} \times \mathbb{N}$ by: $\Uparrow (t,k)$ iff $t = \langle k\rangle v$ or $t = [k/v,l]w$.

Proposition 5.14 (Description of the p_2-normal forms). A p_2-normal form is a term in one of the following forms:

— $\underline{n}$
— λt with $t \in NF(p_2)$
— $(t\ u)$ with $t, u \in NF(p_2)$
— $\langle k\rangle t$ with $t \in NF(p_2)$
— $[i/u,j]\underline{n}$ with $u \in NF(p_2)$
— $[i/u,j]t$ with $t, u \in NF(p_2)$ and there is k such that $k > i$ and $\Uparrow (t,k)$

Proof. By induction on t. □

Proposition 5.15. The p_2-calculus is weakly normalizing.

Proof. Let t be a λ_{ws}-term. We show, by induction on the complexity of t, that $t \in WN(p_2)$. The only difficult case is $t = [i/u,j]v$. By induction, u and v have normal forms u' and v' and we can reduce $t \longrightarrow^* [i/u',j]v'$. We have to show: If $t, u \in NF(p_2)$ then $[i/u,j]t \in WN(p_2)$.

By induction on the complexity of t, we show that there is $t' \in NF(p_2)$ such that:

— $[i/u,j]t \longrightarrow^*_{p_2} t'$
— If $\Uparrow (t,k)$ then $\Uparrow (t', \min(i,k))$

The non trivial cases are:

— If $t = \underline{n}$ then $t' = [i/u,j]\underline{n}$ is a normal form.
— If $t = \langle k\rangle v$:
  - If $i < k$ then $t' = [i/u,j]\langle k\rangle v$ is a normal form and we have $\Uparrow (t', \min(i,k))$.
  - If $i \geq k$ then $[i/u,j]t \longrightarrow \langle k\rangle[i-k/u,j]v$. By induction $[i-k/u,j]v \longrightarrow^* v'$ with $v' \in NF(p_2)$ and so $[i/u,j]t \longrightarrow^* \langle k\rangle v'$ and $\langle k\rangle v' \in NF(p_2)$. Clearly $\Uparrow (t', \min(i,k))$.

— If $t = [k/v,l]w$:
  - If $i < k$ then $t' = [i/u,j]t$ is a normal form and we have $\Uparrow (t', \min(i,k))$.
  - If $k \leq i < k+l$ then $[i/u,j]t \longrightarrow [k/[i-k/u,j]v, l+j-1]w$. By induction $[i-k/u,j]v \longrightarrow^* v'$ with $v' \in NF(p_2)$ and so $[i/u,j]t \longrightarrow^* t' = [k/v', l+j-1]w$. Finally $t' \in NF(p_2)$ and $\Uparrow (t', \min(i,k))$.
  - If $k+l \leq i$ then $[i/u,j]t \longrightarrow [k/[i-k/u,j]v, l][i-l+1/u,j]w$. By induction,

$[i-k/u,j]v \longrightarrow^* v'$ with $v' \in NF(p_2)$, and $[i-l+1/u,j]w \longrightarrow^* w'$ with $w' \in NF(p_2)$. Thus:

$$[i/u,j]t \longrightarrow^*_{p_2} t' = [k/v',l]w'$$

Clearly $\Uparrow (t', \min(i,k))$. The only thing to prove is that t' is a p_2-normal form. w is necessarily of one of the following forms:

- $w = \underline{m}$ and $w' = [i-l+1/u,j]w$ but $k < i-l+1$, and then $t' \in NF(p_2)$.
- $\Uparrow (w,n)$ with $k < n$. By induction hypothesis, $\Uparrow (w', \min(i-l+1,n))$. As $k < i-l+1$, we have $k < \min(n, i-l+1)$ and t' is a p_2-normal form.

□

6. Confluence on open terms

In this section, we show that the λ_{ws}-calculus is confluent on terms with metavariables.

6.1. *The calculus with metavariables*

We enlarge the syntax of terms by allowing metavariables (denoted by $a, b, \ldots$).

Definition 6.1. Λ_{ws_o} is the set of terms with metavariables which are defined by:

$$t ::= \underline{n} \mid a \mid \lambda t \mid (t\ t) \mid \langle k \rangle t \mid [i/t,j]t \quad \text{with} \quad a \text{ a metavariable and } \quad n,i,j,k \in \mathbb{N}$$

Definition 6.2. The λ_{ws_o}-calculus is the reduction on Λ_{ws_o} defined by the rules of the λ_{ws}-calculus. The ws_o-calculus is the reduction defined by the rules of the ws-calculus.

Proposition 6.3. The ws_o-calculus is strongly normalizing.

Proof. It is an immediate consequence of the strong normalization of the ws-calculus. □

6.2. *Confluence of the ws_o-calculus*

Proposition 6.4. The ws_o-calculus is locally confluent.

Proof. By analyzing the critical pairs (see (Guillaume, 1999b)). □

Theorem 6.5. The ws_o-calculus is confluent.

Proof. The ws_o-calculus is locally confluent (proposition 6.4) and strongly normalizing (proposition 6.3). The result follows from Newman's lemma. □

6.3. *Confluence of the λ_{ws_o}-calculus*

In order to show the confluence of the λ_{ws_o}-calculus, we use the interpretation method (Hardin, 1989). This allows to restrict ourselves to confluence on ws_o-normal forms.

Lemma 6.6 (Description of the ws_o-normal forms).
The terms $NF(ws_o)$ are described by the following grammar:
$t ::= u \mid \langle k \rangle u$
$u ::= \underline{n} \mid [i_1/t, j_1] \ldots [i_m/t, j_m]a \mid \lambda t \mid (t\ t)$
with $m \geq 0$, a a metavariable and $k, n, i_1, j_1, \ldots, i_m, j_m \in \mathbb{N}$ such that $i_1 < \ldots < i_m$.

Proof. Trivial. □

We denote by $ws_o(t)$ the ws_o-normal form of t. Remark, in the previous definition, that the term $[i_1/t, j_1] \ldots [i_m/t, j_m]a$ could be written $S\{\!|a|\!\}$ with S a substitutive context, but we have to say that the terms inside the substitutions are ws_o-normal forms.

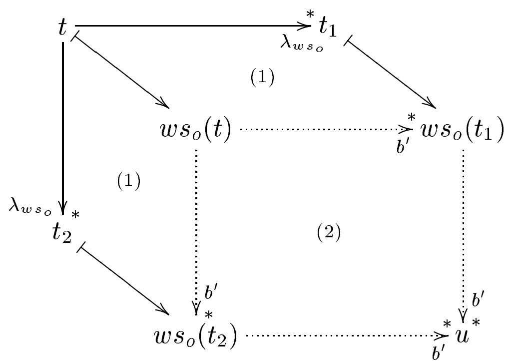
Definition 6.7. On the set $NF(ws_o)$, we define a reduction b' by:
$t \longrightarrow_{b'} u$ iff there is $t' \in \Lambda_{ws}$ such that $t \longrightarrow_b t'$ and $u = ws_o(t')$.

Theorem 6.8. The λ_{ws_o}-calculus is confluent.

Proof. We will check:

(1) If $t \longrightarrow_{\lambda_{ws_o}} u$ then $ws_o(t) \longrightarrow^*_{b'} ws_o(u)$ (proposition 6.15).
(2) b' is confluent (proposition 6.20).

The following diagram (the interpretation method) then gives the confluence of the λ_{ws_o}-calculus.



□

Corollary 6.9. The λ_{ws}-calculus is confluent.

In order to prove (1) and (2) above we need some lemmas. The following definition will simplify the proofs.

Definition 6.10. We define a function $\uparrow$ from Λ_{ws_o} to $\overline{\mathbb{N}}$ by:

— $\uparrow(a) = \infty$,
— $\uparrow([i/u, j]t) = i$,
— $\uparrow(t) = -\infty$ in the other cases.

Remark 6.11. This function allows to write easily the condition under which a substitution can go down in a term: if t, u are ws_o-normal forms then

$$[i/u, j]t \in NF(ws_o) \text{ iff } i < \uparrow(t)$$

Lemma 6.12. Let $t, u \in NF(ws_o)$. Then, $\uparrow(ws_o([i/u,j]t)) = \min(i, \uparrow(t))$.

Proof. Trivial. □

6.3.1. *The calculus on the ws_o-normal forms*

Lemma 6.13. Let $t, u, u' \in NF(ws_o)$ be such that $u \longrightarrow^*_{b'} u'$. Then,

$$ws_o([i/u,j]t) \longrightarrow^*_{b'} ws_o([i/u',j]t)\,.$$

Proof. By an immediate induction on the length of the derivation $u \longrightarrow^*_{b'} u'$, we may assume that $u \longrightarrow_{b'} u'$. The proof is by induction on t. The only interesting case is $t = [k/v,l]w$.

— If $i < k$ then

$$ws_o([i/u,j][k/v,l]w) = [i/u,j][k/v,l]w$$
$$ws_o([i/u',j][k/v,l]w) = [i/u',j][k/v,l]w$$

Thus,

$$ws_o([i/u,j]t) \longrightarrow^*_{b'} ws_o([i/u',j]t)$$

— If $k \le i < k+l$ then

$$ws_o([i/u,j][k/v,l]w) = [k/ws_o([i-k/u,j]v), l+j-1]w$$
$$ws_o([i/u',j][k/v,l]w) = [k/ws_o([i-k/u',j]v), l+j-1]w$$

By induction hypothesis, $ws_o([i-k/u,j]v) \longrightarrow^*_{b'} ws_o([i-k/u',j]v)$ and then

$$ws_o([i/u,j]t) \longrightarrow^*_{b'} ws_o([i/u',j]t)$$

— If $k+l \le i$.

$$ws_o([i/u,j][k/v,l]w) = ws_o([k/[i-k/u,j]v,l][i-l+1/u,j]w)$$

Since $t \in NF(ws_o)$, we have $k < \uparrow(w)$ so $k < \min(\uparrow(w), i+l-1)$. Lemma 6.12 ensures that $k < \uparrow(ws_o([i-l+1/u,j]w))$. We get

$$ws_o([i/u,j][k/v,l]w) = [k/ws_o([i-k/u,j]v),l]ws_o([i-l+1/u,j]w)$$

In the same way,

$$ws_o([i/u',j][k/v,l]w) = [k/ws_o([i-k/u',j]v),l]ws_o([i-l+1/u',j]w)$$

By induction hypothesis, $ws_o([i-k/u,j]v) \longrightarrow^*_{b'} ws_o([i-k/u',j]v)$ and $ws_o([i-l+1/u,j]w) \longrightarrow^*_{b'} ws_o([i-l+1/u',j]w)$. Finally,

$$ws_o([i/u,j]t) \longrightarrow^*_{b'} ws_o([i/u',j]t)\,.$$

□

Lemma 6.14. Let $t, t', u \in NF(ws_o)$ be such that $t \longrightarrow^*_{b'} t'$. Then

$$ws_o([i/u,j]t) \longrightarrow^*_{b'} ws_o([i/u,j]t')$$

Proof. By an immediate induction on the length of the derivation $t \longrightarrow^*_{b'} t'$, we may assume that this reduction is one step. The proof is by induction on t. The only interesting case is when the redex is at the root of t: $t = (\langle k\rangle \lambda v\ w)$ and $t' = ws_o([0/w, k]v)$.

— If $i < k$ then

$$\begin{aligned} ws_o([i/u, j]t) &= (\langle k+j-1\rangle \lambda v\ ws_o([i/u, j]w)) \\ ws_o([i/u, j]t') &= ws_o([i/u, j]ws_o([0/w, k]v)) \\ &= ws_o([i/u, j][0/w, k]v) \\ &= ws_o([0/[i/u, j]w, k+j-1]v) \end{aligned}$$

Thus, $ws_o([i/u, j]t) \longrightarrow^*_{b'} ws_o([i/u, j]t')$.

— If $i \geq k$ then

$$\begin{aligned} ws_o([i/u, j]t) &= (\langle k\rangle \lambda ws_o([i-k+1/u, j])v\ ws_o([i/u, j]w)) \\ ws_o([i/u, j]t') &= ws_o([i/u, j]ws_o([0/w, k]v)) \\ &= ws_o([i/u, j][0/w, k]v) \\ &= ws_o([0/[i/u, j]w, k][i-k+1/u, j]v) \end{aligned}$$

Thus, $ws_o([i/u, j]t) \longrightarrow^*_{b'} ws_o([i/u, j]t')$.

□

Proposition 6.15. If $t \longrightarrow_{\lambda_{ws_o}} u$ then $ws_o(t) \longrightarrow^*_{b'} ws_o(u)$.

Proof. If the reduction $t \longrightarrow_{\lambda_{ws_o}} u$ is a ws_o-reduction, then the uniqueness of ws_o-normal forms gives the result. Assume then that $t \longrightarrow_b u$. The proof is by induction on t:

— If t does not begin with a substitution, the difficult case is when the reduction is at the root, i.e. $t = (\langle k\rangle \lambda v\ w)$ and $u = [0/w, k]v$. We get

$$ws_o(t) = (\langle k\rangle \lambda ws_o(v)\ ws_o(w)) \longrightarrow_{b'} ws_o([0/ws_o(w), k]ws_o(v)) = ws_o(u)$$

— If $t = [i/w, j]v$ and $u = [i/w', j]v$ with $w \longrightarrow_b w'$ then, by the induction hypothesis, $ws_o(w) \longrightarrow^*_{b'} ws_o(w')$. By lemma 6.13, we have

$$ws_o([i/ws_o(w), j]ws_o(v)) \longrightarrow^*_{b'} ws_o([i/ws_o(w'), j]ws_o(v))$$

and then $ws_o(t) \longrightarrow^*_{b'} ws_o(u)$.

— If $t = [i/w, j]v$ and $u = [i/w, j]v'$ with $v \longrightarrow_b v'$ then, by the induction hypothesis, $ws_o(v) \longrightarrow^*_{b'} ws_o(v')$. By lemma 6.14, we have

$$ws_o([i/ws_o(w), j]ws_o(v)) \longrightarrow^*_{b'} ws_o([i/ws_o(w), j]ws_o(v'))$$

and then $ws_o(t) \longrightarrow^*_{b'} ws_o(u)$.

□

6.3.2. *Confluence of the reduction on ws_o-normal forms*

To show the confluence of the b'-reduction, we use the usual method of parallel reductions.

Definition 6.16. We define the *parallel reduction* $\Longrightarrow$ on the set $NF(ws_o)$ by:

— $\underline{n} \Longrightarrow \underline{n}$.

— If $t_1 \Longrightarrow t_2$ then $\lambda t_1 \Longrightarrow \lambda t_2$.
— If $t_1 \Longrightarrow t_2$ and $u_1 \Longrightarrow u_2$ then $(t_1 u_1) \Longrightarrow (t_2 u_2)$.
— If $t_1 \Longrightarrow t_2$ then $\langle k\rangle t_1 \Longrightarrow \langle k\rangle t_2$.
— If $t_k \Longrightarrow u_k$ for $1 \leq k \leq n$ and $i_1 < \ldots < i_n$
then $[i_1/t_1, j_1]\ldots[i_n/t_n, j_n]a \Longrightarrow [i_1/u_1, j_1]\ldots[i_n/u_n, j_n]a$.
— If $t_1 \Longrightarrow t_2$ and $u_1 \Longrightarrow u_2$ then $(\lambda t_1\ u_1) \Longrightarrow ws_o([0/u_2, 0]t_2)$.
— If $t_1 \Longrightarrow t_2$ and $u_1 \Longrightarrow u_2$ then $(\langle k\rangle\lambda t_1\ u_1) \Longrightarrow ws_o([0/u_2, k]t_2)$.

Lemma 6.17. $\Longrightarrow^* = \longrightarrow^*_{b'}$.

Proof. It is easy to see that if $t \longrightarrow_{b'} u$ then $t \Longrightarrow u$. Conversely, assume that $t \Longrightarrow u$. We use an induction on the definition of $t \Longrightarrow u$. The hardest case is the last one:

Let $t = (\langle k\rangle\lambda v\ w)$ and $u = ws_o([0/w', k]v')$ with $v \Longrightarrow v'$ and $w \Longrightarrow w'$. By induction hypothesis, $v \longrightarrow^*_{b'} v'$ and $w \longrightarrow^*_{b'} w'$. Then,

$$t = (\langle k\rangle\lambda v\ w) \longrightarrow^*_{b'} (\langle k\rangle\lambda v'\ w) \longrightarrow^*_{b'} (\langle k\rangle\lambda v'\ w') \longrightarrow_{b'} u = ws_o([0/w', k]v')\,.$$

□

Lemma 6.18. Let $t, u \in NF(ws_o)$. If $t \Longrightarrow t'$ and $u \Longrightarrow u'$, then $ws_o([i/u, j]t) \Longrightarrow ws_o([i/u', j]t')$.

Proof. Let $T = ws_o([i/u, j]t)$ and $T' = ws_o([i/u', j]t')$. We show $T \Longrightarrow T'$ by induction on the definition of $t \Longrightarrow t'$

The difficult cases are:

— $t = [k/v, l]w$ and $t' = [k/v', l]w'$ with $v \Longrightarrow v'$ and $w \Longrightarrow w'$.

- If $i < k$ then $[i/u, j]t$ and $[i/u', j]t'$ are already ws_o-normal forms and so
$$T = [i/u, j][k/v, l]w \Longrightarrow T' = [i/u', j][k/v', l]w'$$
- If $k \leq i < k + l$ then
$$T = [k/ws_o([i-k/u, j]v), l+j-1]w T' = [k/ws_o([i-k/u', j]v'), l+j-1]w'$$
By the induction hypothesis, $ws_o([i-k/u, j]v) \Longrightarrow ws_o([i-k/u', j]v')$ and $T \Longrightarrow T'$.
- If $i \geq k + l$ then
$$T = ws_o([k/[i-k/u, j]v, l][i-l+1/u, j]w)$$
$$T' = ws_o([k/[i-k/u', j]v', l][i-l+1/u', j]w')$$
By the induction hypothesis, $ws_o([i-k/u, j]v) \Longrightarrow ws_o([i-k/u', j]v')$ and $ws_o([i-l+1/u, j]w) \Longrightarrow ws_o([i-l+1/u', j]w')$. Moreover, lemma 6.12 ensures that $\uparrow(ws_o([i-l+1/u, j]w)) = \min(\uparrow(w), i-l+1)$. Since $t \in NF(ws_o)$ we have $k < \uparrow(w)$ and so $k < \min(\uparrow(w), i-l+1)$.
$$T = [k/ws_o([i-k/u, j]v), l]ws_o([i-l+1/u, j]w)$$
$$T' = [k/ws_o([i-k/u', j]v'), l]ws_o([i-l+1/u', j]w')$$
Finally, $T \Longrightarrow T'$.

— $t = (\langle k\rangle\lambda w\ v)$ and $t' = ws_o([0/w', k]v')$ with $v \Longrightarrow v'$ and $w \Longrightarrow w'$.

- If $i < k$

$$T = (\langle k+j-1\rangle\lambda v\ ws_o([i/u, j]w))$$
$$T' = ws_o([i/u', j][0/w', k]v') = ws_o([0/[i/u', j]w', k+j-1]v')$$

  By the induction hypothesis, $ws_o([i/u, j]w) \Longrightarrow ws_o([i/u', j]w')$ so $T \Longrightarrow T'$.

- If $i \geq k$

$$T = (\langle k\rangle\lambda ws_o([i-k+1/u, j]v)\ ws_o([i/u, j]w))$$
$$T' = ws_o([i/u', j][0/w', k]v') = ws_o([0/[i/u', j]w', k][i-k+1/u', j]v')$$

  By the induction hypothesis, $ws_o([i/u, j]w) \Longrightarrow ws_o([i/u', j]w')$ and $T \Longrightarrow T'$.

□

Lemma 6.19. The reduction $\Longrightarrow$ is strongly confluent.

Proof. Let $t_1, t_2, t_3 \in NF(ws_o)$ be such that $t_1 \Longrightarrow t_2$ and $t_1 \Longrightarrow t_3$. We show that there is a term t_4 such that $t_2 \Longrightarrow t_4$ and $t_3 \Longrightarrow t_4$ by induction on the complexity of t_1. The only interesting case is when $t_1 = (\langle k\rangle\lambda u_1\ v_1)$. We consider the form of t_2 and t_3.

— If $t_2 = (\langle k\rangle\lambda u_2\ v_2)$ with $u_1 \Longrightarrow u_2$ and $v_1 \Longrightarrow v_2$.

- If $t_3 = (\langle k\rangle\lambda u_3\ v_3)$ with $u_1 \Longrightarrow u_3$ and $v_1 \Longrightarrow v_3$ the induction hypothesis gives the result.
- If $t_3 = ws_o([0/v_3, k]u_3)$ with $u_1 \Longrightarrow u_3$ and $v_1 \Longrightarrow v_3$ the induction hypothesis ensures that there are u_4 and v_4 such that $u_2 \Longrightarrow u_4$, $u_3 \Longrightarrow u_4$, $v_2 \Longrightarrow v_4$ and $v_3 \Longrightarrow v_4$ and then

$$\begin{array}{ccc}
t_1 = (\langle k\rangle\lambda u_1\ v_1) & \Longrightarrow & t_2 = (\langle k\rangle\lambda u_2\ v_2) \\
\Downarrow & & \Downarrow \\
t_3 = ws_o([0/v_3, k]u_3) & \underset{\text{prev. lemma}}{\Longrightarrow} & t_4 = ws_o([0/v_4, k]u_4)
\end{array}$$

— If $t_2 = ws_o([0/v_2, k]u_2)$ with $u_1 \Longrightarrow u_2$ and $v_1 \Longrightarrow v_2$.

- If $t_3 = (\langle k\rangle\lambda u_3\ v_3)$ with $u_1 \Longrightarrow u_3$ and $v_1 \Longrightarrow v_3$, we conclude as in the previous case.
- If $t_3 = ws_o([0/v_3, k]u_3)$ with $u_1 \Longrightarrow u_3$ and $v_1 \Longrightarrow v_3$ the induction hypothesis ensures that there are u_4 and v_4 such that $u_2 \Longrightarrow u_4$, $u_3 \Longrightarrow u_4$, $v_2 \Longrightarrow v_4$ and $v_3 \Longrightarrow v_4$, then

$$\begin{array}{ccc}
t_1 = (\langle k\rangle\lambda u_1\ v_1) & \Longrightarrow & t_2 = ws_o([0/v_2, k]u_2) \\
\Downarrow & & \Downarrow \text{prev. lemma} \\
t_3 = ws_o([0/v_3, k]u_3) & \underset{\text{prev. lemma}}{\Longrightarrow} & t_4 = ws_o([0/v_4, k]u_4)
\end{array}$$

□

Proposition 6.20. b' is confluent.

Proof. The reduction $\Longrightarrow$ is strongly confluent, therefore the reduction $\longrightarrow^*_{b'}$ is also strongly confluent and then b' is confluent (remark 2.5). $\square$

7. Simulation of the β-reduction

There is a one-one correspondence between one-step reduction in the λ_{db}-calculus and one-step of β-reduction in the λ_w-calculus. In order to show that the λ_{ws}-calculus correctly implements the β-reduction, we give the link with the λ_w-calculus. We show that any reduction of the λ_w-calculus can be done in the λ_{ws}-calculus (cf. proposition 7.3) and that any λ_{ws}-reduction corresponds to a λ_w-reduction on the ws-normal forms (cf. proposition 7.4). In this sense, our calculus has a step by step simulation of β.

Strictly speaking, λ_{ws} does not simulate the λ_{db}-reduction. However, as we already said in subsection 3.5, λ-terms with labels are efficient notations for λ-terms and, when the λ_{ws}-calculus is used as the internal representation of λ-terms (for the implementation of a functional language or a proof assistant), the simulation property we give here is clearly the useful one.

Finally note that $\lambda\zeta$ only simulates big steps of reduction and that the link of the λ_{ws}-calculus with the β-reduction is much simpler than the one of the $SKInT$-calculus: λ-terms trivially are λ_{ws}-terms whereas, in $SKInT$, CPS transformation and abstraction algorithm are necessary to get the translation.

The following property is trivial:

Proposition 7.1. The ws-normal forms are the terms of Λ_w, i.e. they are given by the grammar:

$t ::= u \mid \langle k\rangle u \quad$ with $\; k \in \mathbb{N}$
$u ::= \underline{n} \mid \lambda t \mid (t\; t)$

The set of ws-normal forms will be denoted either by $NF(ws)$ or by Λ_w.

The next lemma gives the relation between the implicit substitution (cf. definition 3.5) and the explicit one.

Lemma 7.2. Let $t, u \in NF(ws)$. Then $\{i/u, j\}t = p([i/u, j]t)$.

Proof. By induction on the complexity of t. $\square$

The following proposition shows that any λ_{ws}-reduction can be simulated in the λ_w-calculus.

Proposition 7.3. Let $t, u \in NF(ws)$. If $t \longrightarrow_{\lambda_w} u$ then $t \longrightarrow^*_{\lambda_{ws}} u$.

Proof. We consider the case $t \longrightarrow_{\beta_2} u$ (the β_1 rule is simpler). Let $t = C\{\!\!\{(\langle k\rangle\lambda v\; w)\}\!\!\}$ and $u = m(C\{\!\!\{\{0/w, k\}t\}\!\!\})$.

— If the context C ends with a label: $C\{\!\!\{\cdot\}\!\!\} = C'\{\!\!\{\langle l\rangle \cdot\}\!\!\}$ then

$$t = C'\{\!\!\{\langle l\rangle(\langle k\rangle\lambda v\; w)\}\!\!\} \longrightarrow_{b_2} C'\{\!\!\{\langle l\rangle[0/w, k]t\}\!\!\} \longrightarrow^*_{ws} C'\{\!\!\{ws(\langle l\rangle[0/w, k]t)\}\!\!\}$$

with remark 4.5 and lemma 7.2,

$$ws(\langle l\rangle[0/w, k]t) = m(p(\langle l\rangle[0/w, k]t)) = m(\langle l\rangle p([0/w, k]t)) = m(\langle l\rangle\{0/w, k\}t)$$

Moreover, as C' cannot end with a label, we have $u = m(C'\{\!\![\langle l\rangle\{0/w,k\}t]\!\!\}) = C'\{\!\![m(\langle l\rangle\{0/w,k\}t)]\!\!\}$ and thus $t \longrightarrow^*_{\lambda_{ws}} u$.

— If the context C does not end with a label:

$$t = C\{\!\![(\langle k\rangle\lambda v\ w)]\!\!\} \longrightarrow_{b_2} C\{\!\![[0/w,k]t]\!\!\} \longrightarrow^*_{ws} C\{\!\![ws([0/w,k]t)]\!\!\}$$

with remark 4.5 and lemma 7.2,

$$ws([0/w,k]t) = m(p([0/w,k]t)) = m(\{0/w,k\}t)$$

Moreover, $u = m(C\{\!\![\{0/w,k\}t]\!\!\}) = C\{\!\![m(\{0/w,k\}t)]\!\!\}$ and thus $t \longrightarrow^*_{\lambda_{ws}} u$.

□

Conversely, we can show that any λ_{ws}-reduction of a term t corresponds to a λ_w-reduction of the ws-normal form of t.

Proposition 7.4. Let $t, u \in \Lambda_{ws}$. If $t \longrightarrow_{\lambda_{ws}} u$ then $ws(t) \longrightarrow^*_{\lambda_w} ws(u)$.

Proof. This is a particular case of proposition 6.15 with terms without metavariables. Just remark that, on terms without metavariables, the reductions λ_w and b' (cf. definition 6.7) are the same. □

8. Preservation of strong normalization

In this section, we give the proof of the preservation of the strong normalization. This property is the hardest one. Since most of the calculi with composition of substitutions fail to have the PSN property, a new technique has to be invented. This technique is inspired by the notion of standard reduction of the λ-calculus.

The labels prevent the loss of information which appears in the λs_e-calculus and in the $\lambda\sigma$-calculus. The rules c_1 and c_2 are exactly the rules needed to obtain both MC and PSN.

As in λs_e, the Melliès counter-example is avoided with the side condition of the interaction rules: a term $[i/u,j][k/v,l]t$ is a redex (rule c_1 or c_2) if and only if $i \geq k$. In λs_e, new rules are added for the propagation of updatings. We have seen, in subsection 2.3, that one of these rules $(\phi\sigma)$ causes the failure of PSN. In λ_{ws}, this rule is useless, since there no need to move updatings down. In this way, λ_{ws} avoids the λs_e counter-example.

The key point of the proof is lemma 8.15. This lemma ensures that it is always possible to do a useful composition to get MC (first point) and that it is never possible to do a useless and dangerous (for PSN) composition (second point). The corresponding lemma would be false for λs_e and $\lambda\sigma$. In other words, unlike λs_e, the substitution have a good behavior: if a term contains a subterm $[i/u,j][k/v,l]t$ with $i < k$ (no possible interaction) then in all future reducts of t it will still be impossible to make these two substitutions interact.

The general idea of the proof is the following: we construct an infinite derivation without composition from an infinite derivation in the λ_{ws}-calculus. This allows to show that we never get artificial terms of the form $[\ldots u \ldots]u$.

In the subsection 8.1, we give the sketch of the proof. Sections 8.2 and 8.3 give the definitions and the main tools used in the proof. The key lemma is proved in section 8.4.

8.1. *Sketch of the proof*

Let $t \in NF(ws)$ be such that $t \in SN(\lambda_w)$. We show that t is strongly normalizable in the λ_{ws}-calculus.

Theorem 8.1. $SN(\lambda_w) \subset SN(\lambda_{ws})$.

For technical reasons, it is easier to work on p-normal forms rather than on ws-normal forms (the p-calculus is the ws-calculus without the mixing rule m). We thus prove the (stronger) result: If $t \in NF(p)$ and $m(t) \in SN(\lambda_w)$ then $t \in SN(\lambda_{ws})$ which is a consequence of the following:

Lemma 8.2 (key lemma). Let $t \in NF(p) \setminus SN(\lambda_{ws})$. There is $u \in NF(p)$ such that $u \notin SN(\lambda_{ws})$ and $m(t) \longrightarrow_{\lambda_w} m(u)$.

Proof of the theorem 8.1. Let $t \in NF(p)$ be such that $m(t) \in SN(\lambda_w)$ and $t \notin SN(\lambda_{ws})$. We can choose t such that the length of the longest λ_w-reduction of $m(t)$ is minimal. The key lemma gives a term u such that the length of the longest λ_w-reduction of $m(u)$ is shorter and thus we get a contradiction. We have proved: If $t \in NF(p)$ and $m(t) \in SN(\lambda_w)$ then $t \in SN(\lambda_{ws})$. The theorem is a particular case of this result with $t \in NF(ws)$ since, for such a t, $m(t) = t$. $\square$

The key lemma is proved by induction. The difficult case is when the head of t is $(\langle k_1 \rangle \dots \langle k_n \rangle \lambda v\ w)$ and v, w as well as all arguments of the head redex are λ_{ws}-strongly normalizable. The term u (given by the lemma) is defined by the following sequence of reductions:

— if $n > 1$, contract the labels $\langle k_j \rangle$,
— reduce the head redex,
— take the p-normal form.

The key point is to show that if t has an infinite λ_{ws}-reduction, then so does u. For the two first steps (contraction of the labels and reduction of the head redex), it is easy to show that infinite reductions are preserved.

For the last step (propagation of the substitution), we use the projection lemma on an extended syntax of the λ_{ws}-calculus. This syntax allows to keep track of the reducts of the substitution created by reduction of the head redex. (subsection 8.3).

8.2. *Definitions*

We give here the definitions which are used in the definition of the term u of the key lemma.

Definition 8.3.
We define particular contexts and terms:

— The *feet* F and the *bodies* B are contexts defined by the grammars:

$$F = \{\!\!\{ \cdot \}\!\!\} \mid \langle k_1 \rangle \ldots \langle k_n \rangle \lambda F$$

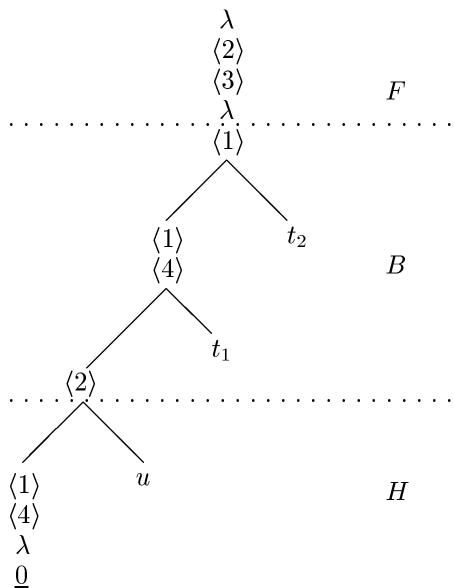
$$B = \{\!\!\{ \cdot \}\!\!\} \mid \langle k \rangle B \mid (B\ t) \text{ with } t \in NF(p)$$

— The *heads* H are terms of the form $\underline{n}$ or $(\langle k_1 \rangle \ldots \langle k_m \rangle \lambda u\ v)$ with $m \geq 0$ and $u, v \in NF(p)$.

Lemma 8.4 (canonical decomposition of the p-normal-forms).
Each term $t \in NF(p)$ has a canonical decomposition $t = F\{\!\!\{B\{\!\!\{H\}\!\!\}\}\!\!\}$.

Proof. By induction on t. □

Example 8.5. Let $t = \lambda\langle 2\rangle\langle 3\rangle\lambda\langle 1\rangle(\langle 1\rangle\langle 4\rangle(\langle 2\rangle(\langle 1\rangle\langle 4\rangle\lambda\underline{0}\ u)\ t_1)\ t_2)$,



The two main points are the following: (1) The interesting reductions of t are the ones of $B\{\!\!\{H\}\!\!\}$. This is due to the fact that a foot is either empty or is a context finishing with a λ. (2) An important information is the level where the substitution created by the reduction of the head redex appears in the term $B\{\!\!\{H\}\!\!\}$. This will be defined (cf. below) as the depth of B.

Definition 8.6.

1 Let B be a body, $Arg(B)$ (the set of *arguments* of B) is defined by:

— If $B = \{\!\!\{ \cdot \}\!\!\}$ then $Arg(B) = \emptyset$.

— If $B = \langle k \rangle B'$ then $Arg(B) = Arg(B')$.

— If $B = (B'\ t)$ then $Arg(B) = Arg(B') \cup \{t\}$.

2 Let B be a body. $|B|$ (the *depth* of B) is defined by:

— $|\{\!\![\cdot]\!\!\}| = 0$.
— $|\langle k\rangle B| = |B| + k$.
— $|(B\ t)| = |B|$.

Let t be the term of example 8.5, then $Arg(B) = \{t_1, t_2\}$ and $|B| = 8$.
The following lemma will be used in the proof of the key lemma.

Lemma 8.7. Let $t \in NF(p)$ and $t', t'' \in \Lambda_{ws}$ be such that $t \longrightarrow_m^* t' \longrightarrow_b t''$. Then $m(t) \longrightarrow_{\lambda_w} m(u)$ where $u = p(t'')$.

Proof. By induction on t. If $t = \langle k_1\rangle \ldots \langle k_n\rangle \lambda v$ or $t = \langle k_1\rangle \ldots \langle k_n\rangle (v\ w)$ and the b-reduction is in v or w, the induction hypothesis immediately gives the result.

The only difficult case is $t = \langle k_1\rangle \ldots \langle k_n\rangle(\langle l_1\rangle \ldots \langle l_m\rangle \lambda v\ w)$ and the b-reduction is the one of the head redex. Then,
$m(t) = \langle k\rangle(\langle l\rangle \lambda m(v)\ m(w))$ with $k = \sum_{i=0}^{n} k_i$ and $l = \sum_{i=0}^{m} l_i$
$t' = \langle k'_1\rangle \ldots \langle k'_{n'}\rangle(\langle l\rangle \lambda v'\ w')$ with $v \longrightarrow_m^* v'$, $w \longrightarrow_m^* w'$ and $k = \sum_{i=0}^{n'} k'_i$.
$t'' = \langle k'_1\rangle \ldots \langle k'_{n'}\rangle[0/w', l]v'$.
$m(u) = m(p(t'')) = ws(t'') = ws(\langle k\rangle[0/w', l]v')$.
Finally, $m(t) \longrightarrow_{\lambda_w} ws(\langle k\rangle[0/w, l]v) = m(u)$ (because the ws-calculus is confluent and normalizing). □

8.3. *Preservation of infinite reductions by propagation*

The goal of this subsection is to prove the following lemma; it is the hardest part of the proof. The meaning of this lemma is that the infinite reduction is preserved by the propagation of the head substitution.

Lemma 8.8. Let $t = B\{\!\![[0/w, l]v]\!\!\}$ where $v, w \in NF(p)$. Assume that v, w and the arguments of B are λ_{ws}-strongly normalizable. If t has an infinite λ_{ws}-reduction then $p(t)$ also has an infinite λ_{ws}-reduction.

The idea of the proof is the following: let $u = p(t)$. In order to translate the reduction $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$ into a reduction $u \longrightarrow u_1 \longrightarrow u_2 \longrightarrow \ldots$, we will tag the reducts of the substitution $[0/w, l]$ and write them $[\![0/w, l]\!]$. Then, in any reduct of t, there are two kinds of substitutions: the tagged ones (denoted $[\![\ldots]\!]$) which are reducts of the head substitution of t and the other ones (denoted $[\ldots]$) which are created during the reduction.

The key point (which allows to construct the derivation of u) consists in proving the following properties of the t_i: they ensure that, in each t_i we can move down the substitutions $[\![\ldots]\!]$ without moving the substitutions $[\ldots]$ and thus define u_i as the "normal form of t_i by tagged propagation".

— If a subterm is $[\![i/w', j]\!]v'$ then v' and w' contain no substitution $[\![\ldots]\!]$.
— Substitutions $[\![\ldots]\!]$ are always "higher" than the $[\ldots]$, i.e. if the subterm is $[\![\ldots]\!][\ldots]w$ then we can always compose the substitutions. Conversely, if the subterm is $[\ldots][\![\ldots]\!]w$, the composition is never possible.

— If a subterm is $[\![i/w', j]\!]v'$ then w' is strongly normalizable.

The first property comes from the syntax of the $\lambda^{\diamond}_{ws}$-calculus (cf. 8.3.1). The two others are proved in subsection 8.3.2 and are derived from the notion of well-tagged terms. Finally, it will remain to check that the terms u_i give an infinite λ_{ws}-reduction of u.

8.3.1. *The tagged reductions*

Definition 8.9 (The $\lambda^{\diamond}_{ws}$-calculus). The set of terms of the $\lambda^{\diamond}_{ws}$-calculus (denoted by $\Lambda^{\diamond}_{ws}$) is defined by:

$$t = \underline{n} \mid \lambda t \mid (t\ t) \mid \langle k\rangle t \mid [i/t, j]t \mid [\![i/u, j]\!]v \quad \text{with} \quad n, i, j, k \in \mathbb{N} \quad \text{and} \quad u, v \in \Lambda_{ws}.$$

The rules of the $\lambda^{\diamond}_{ws}$-calculus are those of the λ_{ws}-calculus with the additional rules:

$$\begin{array}{llcll}
l_\diamond & [\![i/u, j]\!]\lambda t & \longrightarrow & \lambda[\![i+1/u, j]\!]t & \\
a_\diamond & [\![i/u, j]\!](t\ v) & \longrightarrow & ([\![i/u, j]\!]t\ [\![i/u, j]\!]v) & \\
e_{1\diamond} & [\![i/u, j]\!]\langle k\rangle t & \longrightarrow & \langle k+j-1\rangle t & i < k \\
e_{2\diamond} & [\![i/u, j]\!]\langle k\rangle t & \longrightarrow & \langle k\rangle[\![i-k/u, j]\!]t & k \le i \\
n_{1\diamond} & [\![i/u, j]\!]\underline{n} & \longrightarrow & \underline{n} & n < i \\
n_{2\diamond} & [\![i/u, j]\!]\underline{n} & \longrightarrow & \langle i\rangle u & n = i \\
n_{3\diamond} & [\![i/u, j]\!]\underline{n} & \longrightarrow & \underline{n+j-1} & i < n \\
c_{1\diamond} & [\![i/u, j]\!][k/v, l]t & \longrightarrow & [k/[\![i-k/u, j]\!]v, l+j-1]t & k \le i < k+l \\
c_{2\diamond} & [\![i/u, j]\!][k/v, l]t & \longrightarrow & [k/[\![i-k/u, j]\!]v, l][\![i-l+1/u, j]\!]t & k+l \le i
\end{array}$$

It is easy to check that the set $\Lambda^{\diamond}_{ws}$ is closed under this reduction: the only constraint imposed by the syntax is that the subterms under or inside a tagged substitution are λ_{ws}-terms (i.e. without tagged substitution). This constraint is clearly preserved by the new rules.

Definition 8.10. The $\diamond$-calculus is the calculus on the set $\Lambda^{\diamond}_{ws}$ which contains the rules of propagation of tagged substitutions: $l_\diamond$, $a_\diamond$, $e_\diamond$, $n_\diamond$ and $c_\diamond$.

Remark 8.11. Note that the following rules

$$\begin{array}{llcll}
c'_{1\diamond} & [i/u, j][\![k/v, l]\!]t & \longrightarrow & [\![k/[i-k/u, j]v, l+j-1]\!]t & k \le i < k+l \\
c'_{2\diamond} & [i/u, j][\![k/v, l]\!]t & \longrightarrow & [\![k/[i-k/u, j]v, l]\!][i-l+1/u, j]t & k+l \le i
\end{array}$$

which would be natural in a general framework are missing in the $\lambda^{\diamond}_{ws}$-calculus. We will have to consider only well-tagged terms (cf. below) of the $\lambda^{\diamond}_{ws}$-calculus and these terms do not contain any $c'_{1\diamond}$-redex or $c'_{2\diamond}$-redex. These rules are thus useless.

8.3.2. *The well-tagged terms*

Here, we formalize the following intuitive fact: in the terms that we are interested in, the tagged substitutions are always higher than the others. We actually define a more general property which is preserved by reduction.

The relation $\mathcal{H}$ between a term (with tagged substitutions) and an integer means that any untagged substitution has a small enough index if a tagged substitution occurs below. The integer gives the depth where the tagged substitution (if any) is in the term.

The relation $\mathcal{B}$ between a term (without tagged substitutions) and an integer means that any untagged substitution which occurs under a tagged one has a small enough index, allowing thus the tagged substitution to be propagated.

Definition 8.12. We define the binary relations by:

— $\mathcal{B}$ on $\Lambda_{ws} \times \mathbb{N}$

- $\mathcal{B}(\underline{n}, m)$
- $\mathcal{B}(\lambda u, m)$ iff $\mathcal{B}(u, m+1)$
- $\mathcal{B}((u\ v), m)$ iff $\mathcal{B}(u, m)$ and $\mathcal{B}(v, m)$
- $\mathcal{B}(\langle i \rangle u, m)$ iff $\begin{cases} i \leq m \text{ and } \mathcal{B}(u, m-i) \\ \text{or} \\ i > m \end{cases}$
- $\mathcal{B}([i/u, j]v, m)$ iff $\begin{cases} i \leq m < i+j \text{ and } \mathcal{B}(u, m-i) \\ \text{or} \\ i+j \leq m \text{ and } \mathcal{B}(u, m-i) \text{ and } \mathcal{B}(v, m-j+1) \end{cases}$

— $\mathcal{H}$ on $\Lambda_{ws}^{\diamond} \times \mathbb{N}$

- $\mathcal{H}(\underline{n}, m)$
- $\mathcal{H}(\lambda u, m)$ iff $\mathcal{H}(u, m+1)$
- $\mathcal{H}((u\ v), m)$ iff $\mathcal{H}(u, m)$ and $\mathcal{H}(v, m)$
- $\mathcal{H}(\langle i \rangle u, m)$ iff $\begin{cases} i \leq m \text{ and } \mathcal{H}(u, m-i) \\ \text{or} \\ i > m \text{ and } u \in \Lambda_{ws} \end{cases}$
- $\mathcal{H}([i/u, j]v, m)$ iff $\begin{cases} m < i \text{ and } u \in \Lambda_{ws} \text{ and } v \in \Lambda_{ws} \\ \text{or} \\ i \leq m < i+j \text{ and } \mathcal{H}(u, m-i) \text{ and } v \in \Lambda_{ws} \\ \text{or} \\ i+j \leq m \text{ and } \mathcal{H}(u, m-i) \text{ and } \mathcal{H}(v, m-j+1) \end{cases}$
- $\mathcal{H}([\![i/u, j]\!]v, m)$ iff $i = m$, $u \in \Lambda_{ws}$, $u \in SN(\lambda_{ws})$ and $\mathcal{B}(v, m)$

Definition 8.13. A term $t \in \Lambda_{ws}^{\diamond}$ is *well-tagged* if there is an integer m such that $\mathcal{H}(t, m)$. The set of well-tagged terms is denoted by WT.

Remark 8.14. The following facts are immediate (by induction on t):

— If $t \in NF(p)$ (i.e. t contains no substitution) then, for all $m \in \mathbb{N}$, we have $\mathcal{B}(t, m)$.

— If $t \in \Lambda_{ws}$ (i.e. t contains no tagged substitution) then, for all $m \in \mathbb{N}$, we have $\mathcal{H}(t, m)$.

— If $t \in WT$ then, for all u subterm of t, we have $u \in WT$.

The following lemma gives the desired properties of well-tagged terms.

Lemma 8.15. Let t be a well-tagged term.

1 If $[\![i/u, j]\!][k/v, l]w$ is a subterm of t, then $i \geq k$ (i.e. the subterm is a $c_{1\diamond}$-redex or a $c_{2\diamond}$-redex).

2 If $[i/u, j][\![k/v, l]\!]w$ is a subterm of t, then $i < k$ (i.e. there is no $c'_{1\diamond}$-redex or $c'_{2\diamond}$-redex (cf. remark 8.11)).

Proof.

1 Let $t' = [\![i/u, j]\!][k/v, l]w$ be the subterm. This is a well-tagged term (remark 8.14(3)) therefore there is an integer m such that $\mathcal{H}(t', m)$. The definition of $\mathcal{H}$ implies that $m = i$ and $\mathcal{B}([k/v, l]w, i)$. The definition of $\mathcal{B}$ implies $k \leq i$.

2 Let $t' = [i/u, j][\![k/v, l]\!]w$ be the subterm. There is an integer m such that $\mathcal{H}(t', m)$. Since $[\![k/v, l]\!]w \notin \Lambda_{ws}$, we have $m \geq i + j$ and $\mathcal{H}([\![k/v, l]\!]w, m - j + 1)$, and thus $k = m - j + 1$. Finally, $k = m - j + 1 > i$.

□

Proposition 8.16. WT is closed by $\lambda^{\diamond}_{ws}$-reduction.

Proof. The proof is not difficult but tedious. We first prove that if $t \longrightarrow_{\lambda_{ws}} u$ and $\mathcal{B}(t, m)$ then $\mathcal{B}(u, m)$. We may assume that the reduction is at the root of t. We consider each rule of the λ_{ws}-calculus. The proposition is a consequence of the fact that if $t \longrightarrow_{\lambda^{\diamond}_{ws}} u$ and $\mathcal{H}(t, m)$ then $\mathcal{H}(u, m)$. This is proved by induction on t using the previous fact. Again we may assume that the reduction is at the root, and we consider each rule of the $\lambda^{\diamond}_{ws}$-calculus. The complete proof is given in the annex of (Guillaume, 1999b). □

Lemma 8.17. The $\diamond$-calculus is confluent and strongly normalizable on the set of well-tagged terms. Let $\diamond(t)$ denote the normal form of t for the $\diamond$-calculus.

Proof. The $\diamond$-calculus is locally confluent because it has no critical pairs. The strong normalization of the $\diamond$-calculus is a trivial consequence of the strong normalization of the ws-calculus. □

Proposition 8.18. Let t be a well-tagged term. Then $\diamond(t) \in \Lambda_{ws}$.

Proof. If t is a well-tagged term then $\diamond(t)$ also is one (proposition 8.16). If $\diamond(t) \notin \Lambda_{ws}$ then it contains a tagged substitution and the lemma 8.15(1) ensures that we can move down this substitution. This contradicts the fact that $\diamond(t)$ is a $\diamond$-normal form. □

8.3.3. *The projection*

We show that an infinite $\lambda^{\diamond}_{ws}$-reduction of a well-tagged term t, gives an infinite λ_{ws}-reduction of $\diamond(t)$. This is done by showing that the relations R_1 and R_2 defined below satisfy the hypothesis of the projection lemma (lemma 2.7).

— R_1: the $\diamond$-reductions and the reductions inside tagged substitutions (i.e. $[\![i/u, j]\!]t \longrightarrow [\![i/u', j]\!]t$ with $u \longrightarrow_{\lambda_{ws}} u'$).

— R_2: the other reductions, i.e. the λ_{ws}-rules used outside a tagged substitution.

Lemma 8.19. $WT \subset SN(R_1)$.

Proof. The measure $||\cdot||$ is defined on well-tagged terms as follows. $\lg(u)$ denotes the length of the longest λ_{ws}-derivation of u (which exists since any term inside a tagged substitution is strongly normalizable). Note that this measure is not the same as the one in proposition 5.12.

— $||\underline{n}|| = 0$
— $||\lambda t|| = ||t||$
— $||(t\ u)|| = ||t|| + ||u||$
— $||\langle k\rangle t|| = ||t||$
— $||[i/u, j]t|| = ||t|| + ||u||$
— $||[\![i/u, j]\!]t|| = \mathrm{cxty}(t)(1 + \lg(u))$

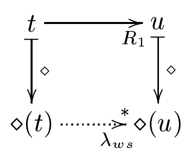
We have to show that if $t \longrightarrow_{\diamond} u$ then $||t|| > ||u||$. By induction on t, we may assume that the reduction is at the root. For the rules $e_{1\diamond}$, $n_{1\diamond}$, $n_{2\diamond}$ and $n_{3\diamond}$, note that if a term has no tagged substitution its measure is 0. For the other rules, the verification is immediate.

We also have to show that the measure decreases by reduction inside tagged substitutions. By induction on t, we may assume that the reduction is $[\![i/u, j]\!]t \longrightarrow [\![i/u', j]\!]t$ with $u \longrightarrow_{\lambda_{ws}} u'$. We have $\lg(u') < \lg(u)$ hence (since $\mathrm{cxty}(t) \geq 1$):

$$||[\![i/u, j]\!]t|| = \mathrm{cxty}(t)(1 + \lg(u)) > \mathrm{cxty}(t)(1 + \lg(u')) = ||[\![i/u', j]\!]t||\,.$$

□

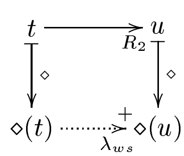
Lemma 8.20. Let t be a well-tagged term. If $t \longrightarrow_{R_1} u$ then $\diamond(t) \longrightarrow^*_{\lambda_{ws}} \diamond(u)$.



Proof. If the reduction $t \longrightarrow_{R_1} u$ is a $\diamond$-reduction then, by uniqueness of the $\diamond$-normal form, $\diamond(t) = \diamond(u)$.

If the reduction is inside a tagged substitution, we use induction on t. The difficult case is $t = [\![i/v, j]\!]w$ and $u = [\![i/v', j]\!]w$ with $v \longrightarrow_{\lambda_{ws}} v'$. An induction on w gives $\diamond(t) \longrightarrow^*_{\lambda_{ws}} \diamond(u)$. □

Lemma 8.21. Let t be a well-tagged term. If $t \longrightarrow_{R_2} u$ then $\diamond(t) \longrightarrow^+_{\lambda_{ws}} \diamond(u)$.



Proof. This proof, by induction on t, is easy but tedious. The difficult case is $t =$

$[\![i/v,j]\!]w$ and $u = [\![i/v,j]\!]w'$ with $w \longrightarrow_{R_2} w'$. By an induction on w we may assume that the reduction is at the root of w. We then have to consider each rule of λ_{ws}-calculus. This proof has been checked by a Caml Program and is given in the annex of (Guillaume, 1999b). □

Proposition 8.22. Let $t \in \Lambda^{\diamond}_{ws}$ be a well-tagged term. If t has an infinite $\lambda^{\diamond}_{ws}$-reduction then $\diamond(t)$ has an infinite λ_{ws}-reduction.

Proof. The previous lemmas prove the hypothesis of the projection lemma. □

We are now ready to finish the proof of the main result of this subsection.

Proof of lemma 8.8 Let $t = B\{\!|[0/w,l]v|\!\}$ with $v, w \in NF(p)$. Assume that v, w and the arguments of B are λ_{ws}-strongly normalizable but t is not strongly normalizable. Let $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$ be an infinite reduction of t and let t'_i be t_i where the residue of $[\![0/w,l]\!]$ has been tagged.

— $t' = B\{\!|[\![0/w,l]\!]v|\!\}$ is a well-tagged term: by induction on B, we prove that $\mathcal{H}(t', |B|)$ (cf. definitions 8.6 and 8.12)
  - If $B = \{\!|\cdot|\!\}$, $t' = [\![0/w,l]\!]v$ and $w \in SN(\lambda_{ws})$. Moreover $v \in NF(p)$. By remark 8.14(1), we have $\mathcal{B}(v,0)$. Finally $\mathcal{H}(t',0)$.
  - if $B = \langle k\rangle B'$: by the induction hypothesis $\mathcal{H}(B'\{\!|[\![0/w,l]\!]v|\!\}, |B'|)$ and so $\mathcal{H}(\langle k\rangle B'\{\!|[\![0/w,l]\!]v|\!\}, |B'|+k)$, i.e. $\mathcal{H}(t', |B|)$.
  - If $B = (B'w')$ with $w' \in \Lambda_{ws}$: by induction, $\mathcal{H}(B'\{\!|[\![0/w,l]\!]v|\!\}, |B'|)$. The remark 8.14(2) gives $\mathcal{H}(w', |B|)$. Finally, $\mathcal{H}(t', |B|)$.

— $t' \longrightarrow t'_1 \longrightarrow t'_2 \longrightarrow \ldots$ is an infinite $\lambda^{\diamond}_{ws}$-reduction of t': proposition 8.16 and lemma 8.15 that each t'_i is well-tagged and that the reduction $t'_i \longrightarrow t'_{i+1}$ is always possible (no $c'_{1\diamond}$ or $c'_{2\diamond}$ redex), respectively.
  the proposition 8.16 ensures that each t'_i is well-tagged and lemma 8.15 ensures that the reduction $t'_i \longrightarrow t'_{i+1}$ is always possible (no $c'_{1\diamond}$ or $c'_{2\diamond}$ redex).

— $\diamond(t') = p(t)$: t' has no untagged substitutions. The reduction from t to $p(t)$ can be translated into a reduction from t' to $\diamond(t')$ by using rules $l_\diamond$, $a_\diamond$, $e_\diamond$ and $n_\diamond$ instead of l, a, e and n. Thus $\diamond(t')$ and $p(t)$ differ only by the character of their substitutions (tagged or not). Since they have no substitutions, $\diamond(t') = p(t)$.

— $p(t)$ has an infinite reduction. t' is well-tagged and has an infinite $\lambda^{\diamond}_{ws}$-reduction. Proposition 8.22 gives an infinite λ_{ws}-reduction of $\diamond(t') = p(t)$.

□

8.4. *Proof of the key lemma*

The proof of the key lemma finishes the proof of theorem 8.1:
Lemma 8.2 Let $t \in NF(p) \setminus SN(\lambda_{ws})$. There is $u \in NF(p)$ such that $u \notin SN(\lambda_{ws})$ and $m(t) \longrightarrow_{\lambda_w} m(u)$.

Proof. We prove, by induction on t, that there is a term u such that:

— $u \notin SN(\lambda_{ws})$,

— there are $t', t'' \in \Lambda_{ws}$ such that $t \longrightarrow_m^* t' \longrightarrow_b t''$ and $u = p(t'')$.

The result follows then immediately from lemma 8.7.

1) If t has a proper subterm v which is not λ_{ws}-strongly normalizing then $t = C\{\!\!\{v\}\!\!\}$ and there is a term $w \notin SN(\lambda_{ws})$ such that $w = p(v'')$ and $v \longrightarrow_m^* v' \longrightarrow_b v''$. Let $u = C\{\!\!\{w\}\!\!\}$. Since C has no substitutions, $p(u) = C\{\!\!\{p(w)\}\!\!\}$.

2) Else, Every proper subterm of t is λ_{ws}-strongly normalizable. Let $t = F\{\!\!\{B\{\!\!\{H\}\!\!\}\}\!\!\}$. F is empty since (if not, $B\{\!\!\{H\}\!\!\}$ would be a non λ_{ws}-strongly normalizable proper subterm of t). H is not a de Bruijn index since, otherwise, t would be strongly normalizable. Thus $t = B\{\!\!\{(\langle k_1\rangle \dots \langle k_n\rangle \lambda v\ w)\}\!\!\}$. Let $k = \sum_{i=0}^{n} k_i$ and

$$t' = B\{\!\!\{(\langle k\rangle \lambda v\ w)\}\!\!\} \qquad t'' = B\{\!\!\{[0/w, k]v\}\!\!\} \qquad u = p(B\{\!\!\{[0/w, k]v\}\!\!\})$$

By construction: $t \longrightarrow_m^* t' \longrightarrow_b t''$ and $u = p(t'')$. It remains to prove that u has an infinite λ_{ws}-reduction.

Since every subterm is strongly normalizable, any infinite reduction of t must reduce the head redex. The infinite reduction of t looks like:

$$t \longrightarrow^* B'\{\!\!\{(\langle k\rangle \lambda v'\ w')\}\!\!\} \longrightarrow B'\{\!\!\{[0/w', k]v'\}\!\!\} \longrightarrow \dots$$

And thus t'' has an infinite reduction:

$$t'' = B\{\!\!\{[0/w, k]v\}\!\!\} \longrightarrow^* B'\{\!\!\{[0/w', k]v'\}\!\!\} \longrightarrow \dots$$

Lemma 8.8 ensures that $u = p(t'')$ has an infinite reduction. □

9. Conclusion

The counter-examples to the preservation of strong normalization of the $\lambda\sigma$-calculus and the λs_e-calculus led us to introduce the λ_w-calculus: a new presentation of the β-reduction.

We then derived a calculus with explicit substitutions satisfying: step by step simulation of β, confluence on terms with metavariables and preservation of strong normalization.

The simulation property of our calculus is not exactly the expected one, however, we believe that the idea of keeping updating functions in terms rather than pushing them down is one of the interesting points of our calculus.

This calculus is the first (together with $SKInT$ of Goubault and Goguen) to answer positively the open question on the existence of such a calculus. We believe that the link of our calculus with De Bruijn calculus is much simpler than the one of the $SKInT$-calculus.

We leave for future work the study of other systems of types for the λ_{ws}-calculus. The implementation of this calculus would also be interesting in order to measure the efficiency of the use of labels.

Acknowledgements

We would like to thank Pierre-Louis Curien, Délia Kesner, Yves Lafont and the anonymous referees for their helpful comments on this work.

References

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

Benaissa, Z.-E.-A., Briaud, D., Lescanne, P., and Rouyer-Degli, J. (1996). $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722.

David, R. (1994). The inf function in the system F. *Theoretical Computer Science*, 135:423–431.

David, R. and Guillaume, B. (1999). The λ_l-calculus (extended abstract). In *Proceedings of The Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (WESTAPP'99)*, Trento.

Di Cosmo, R. and Kesner, D. (1997). Strong normalization of explicit substitutions via cut elimination in proof nets. In *Proceedings of the 12^{th} Annual IEEE Symposium on Logic In Computer Science (LICS)*, Warsaw.

Di Cosmo, R., Kesner, D., and Polonovsky, E. (2000). Proof Nets and Explicit Substitutions. In *Foundations of Software Science and Computation Structures (FOSSACS)*. To appear.

Ferreira, M.-C.-F., Kesner, D., and Puel, L. (1996). λ-calculi with explicit substitutions and composition which preserve strong normalization. *Proceedings of Algebraic and Logic Programming 96* in *Lecture Notes in Computer Science*, 1139:284–298.

Goguen, H. and Goubault-Larrecq, J. (1999). Sequent combinators: A hilbert system for the lambda calculus. *Mathematical Structures in Computer Science, special issue in honor of Sir Roger Hindley.*

Guillaume, B. (1999a). The λs_e-calculus does not preserve strong normalisation. *To appear in Journal of Functional Programming.*

Guillaume, B. (1999b). *Un calcul de substitutions avec étiquettes.* PhD thesis, Université de Savoie. (URL : `http://www.lama.univ-savoie.fr/users/GUILLAUME`).

Hardin, T. (1989). Confluence results for the pure strong categorical logic CCL : λ-calculi as subsystems of CCL. *Theoretical Computer Science*, 65(2):291–342.

Kamareddine, F. and Ríos, A. (1995a). A λ-calculus à la de Bruijn with explicit substitutions. *Proceedings of the 7th international symposium on Programming Languages: Implementations, Logics and Programs, PLILP '95* in *Lecture Notes in Computer Science*, 982:45–62.

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

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

Kamareddine, F. and Ríos, A. (1998). Bridging de bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the IGPL*, 6(6):843–874.

Klop, J. (1992). Term Rewriting Systems. In Abramsky, S., Gabbay, D., and Maibaum, T., editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press.

Melliès, P.-A. (1995). Typed λ-calculi with explicit substitutions may not terminate. *Proceedings of Typed Lambda Calculi and Applications 95* in *Lecture Notes in Computer Science*, 902:328–334.

Muñoz, C. (1996). Confluence and preservation of strong normalisation in an explicit substitutions calculus. *Proceeddings of LICS'96*, pages 440–447.

Muñoz, C. (1997). *Un calcul de substitutions pour la représentation de preuves partielles en théorie des types*. PhD thesis, Université Paris VII.

Zantema, H. (1998). The $\sigma\sigma$-rule terminates. Personnal communication.