Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 5668

On strong normalisation of explicit substitution calculi

Frédéric Lang
Pierre Lescanne

August 31, 1999

Research Report N° RR 1999-37

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

On strong normalisation of explicit substitution calculi

Frédéric Lang
Pierre Lescanne

August 31, 1999

Abstract

In this paper, we present an attempt to build a calculus of explicit substitution expected to be confluent on open terms, to preserve strong normalisation and to simulate one step β-reduction. We show why our attempt failed and we explain how we found a counter-example to the strong normalisation or termination of the substitution calculus. As a consequence, we provide also a counter-example to the strong normalisation of another calculus, namely τ (the substitution calculus of $\lambda\tau$) of Riòs, for which the problem was open.

Keywords: Explicit Substitution, Strong Normalisation, Confluence, Termination, Lambda Calculus

Résumé

Dans cet article, nous rendons compte d'une tentative pour construire un calcul de substitutions explicite sensé être confluent sur les termes ouverts, préserver la forte normalisation et simuler la β-réduction en une seule étape. Nous montrons pourquoi notre tentative a échoué et nous expliquons comment nous avons trouvé un contrexemple à la terminaison du calcul de substitutions. Comme conséquence nous exhibons un contrexemple à la forte normalisation d'un autre calcul à savoir le calcul τ (le calcul de substitution de $\lambda\tau$) de Riòs, pour lequel le problème était ouvert.

Mots-clés: Substitution Explicite, Normalisation Forte, Confluence, Terminaison ,Lambda Calcul

1 Introduction

This paper deals with *λ-calculi with explicit substitution i.e.*, first order calculi which simulate the β-reduction of the λ-calculus. Many such calculi have been proposed and studied and much attention has been done on *confluence*, *one step simulation of the β-reduction*, and *preservation of strong β-normalisation*.

Confluence. Given a relation R on a set S, confluence ensures that for every element a of S, if a rewrites to b in one or more steps and if a rewrites to c in one or more steps, then there exists a d such that both b and c rewrite to d in one or more steps. In the setting of term rewriting systems, one distinguishes between confluence on closed terms when S is the set of terms without first-order variables and confluence on open terms when S is the set of patterns of terms or terms with meta-variables or unbound variables. The first is called *ground confluence* and the second is called *meta-confluence*.

One step simulation. λ-calculi of explicit substitution are meant to simulate λ-calculus. There are two ways to consider it. Either the λ-calculus of explicit substitution is used to compute the same normal forms as the λ-calculus, in other words both calculi lead to the same normal form. Or one has a calculus of explicit substitution which simulates faithfully the λ-calculus, in the sense that each reduction in the λ-calculus can be simulated by reduction in the λ-calculus of explicit substitution. Usually this is done in a few steps, namely one step, which creates an explicit substitution, followed by steps which get rid of this substitution. If the system simulates λ-calculus in the second sense, one says that it fulfils *one step simulation.*

Preservation of strong β-normalisation. An explicit substitution calculus adds new possible reductions w.r.t. the λ-calculus, hence it may be the case that a term which has no infinite derivation in the λ-calculus has ones in the λ-calculus with explicit substitution. If this is never the case the calculus is said to *preserve strong β-normalisation* of the λ-calculus.

To make short, let us call ground confluence GC, meta-confluence MC, one step simulation $\mathsf{1SIM}$, and preservation of strong normalisation PSN.

People are interested in these properties, because they insure that the whole behaviour of the λ-calculus is preserved. The first calculus of explicit substitution is the $\lambda C\xi\phi$ of de Bruijn [dB78], of which [BBLRD96] has a presentation in today notations. But it is somewhat traditional to set the beginning of explicit substitutions with [ACCL91] where the authors define a one step simulating calculi they call $\lambda\sigma$. They are mainly interested on the meta-confluence issue, but they are not aware of the PSN issue. Actually, the calculus $\lambda\sigma_{\Uparrow}$ [CHL96], a direct descendant of $\lambda\sigma$, is the first meta-confluent and one step simulating calculus of explicit substitution to date. Later on, Melliès [Mel95] exhibited a simply typable λ-term which leads to an infinite derivation in $\lambda\sigma_{\Uparrow}$. Guillaume [Gui98, Gui99] did the same for λs_e [KR97], which was expected to solve the problem of $\lambda\sigma_{\Uparrow}$. Other calculi [BBLRD96, BR95, KR95, LRD95, FKP99], have $\mathsf{1SIM}$, PSN and, GC, but not MC. These calculi define, in some sense, sub-relations of the previous ones, restricted enough to preserve strong normalisation, not too much so to keep one step simulation, but too much to preserve confluence on open terms. In another direction, Muñoz [Muñ96] defined a restricted relation as a calculus which has PSN and MC, but not 1SIM.

We worked on the problem of finding a calculus which would enjoy 1SIM, MC and PSN. For this we defined our own calculus. We failed in our endeavour, but in that process we found a counter-example to the strong normalisation of its substitution calculus. This was not a real failure, since we immediately adapted this counter example to the calculus $\lambda\tau$ of Ríos [Río93] (also presented in [Les94]) which was at that time a challenger for 1SIM, PSN and MC. As this approach is instructive, we feel it is worth to be presented.

Plan of the paper. In Section 2, we present our calculus $\lambda\upsilon c$, obtained by joining the critical pairs of the calculus $\lambda\upsilon$ [Les94, BBLRD96]. We show in Section 3, using an abstraction presented as a kid game, that a subsystem of two rules extracted from $\lambda\upsilon c$ is not strongly normalizing, whereas it should to preserve strong normalisation. In Section 4 we adapt the counter example to $\lambda\tau$. We conclude in Section 5.

Preliminary Definitions. Given a relation $\to$, we denote by $\to^+$ its transitive closure and by $\twoheadrightarrow$ its reflexive and transitive closure. In this paper we mention calculi called $\lambda\,\square$ where $\square$ can be filled by any of υ, υc, τ, $\sigma_{\Uparrow}$, s_e, *etc.* We adopt the convention to designate by $\square$ the substitution calculus included in $\lambda\,\square$ *i.e.*, the calculus induced by all rules but the one which introduces the substitution by contraction of a β-redex, namely (B-$\square$).

2 Joining the critical pairs of $\lambda\upsilon$

The calculus $\lambda\upsilon$ (*lambda upsilon*) [Les94, BBLRD96] is a λ-calculus with explicit substitution. Unlike the λ-calculus, the substitution is part of the syntax, here written between brackets. Figure 1 presents $\lambda\upsilon$. It was designed as a *minimal* calculus *i.e.*, a calculus in which every operator and rule is introduced by necessity to get a calculus simulating (one step) β-reduction. These operators are $/$, $\uparrow$, and $\Uparrow$ (read respectively *slash*, *shift*, and *lift*). Roughly speaking, $/$ is introduced by (B-υ), where $a[b/]$ means that the variable denoted by $\underline{0}$ is to be substituted by b in a. The operator $\Uparrow$ is introduced by (Lambda-υ), to express that some indexes in the substitution will have to be incremented due to the propagation under a binder. At last, the operator $\uparrow$ is introduced by (RVarlift-υ), to actually increment the free indexes of a term. Hence, both the α-renaming and the substitution, usually implicit in the λ-calculus, are here performed explicitly.

The calculus $\lambda\upsilon$ has 1SIM, PSN, and GC. However, it is not confluent on open terms, as shown in Figure 2 by the non-joinable critical pair between rules (B-υ) and (App-υ). A system which has no such critical pair is said *locally confluent.* One technique to get a confluent system from a non-confluent one is the so-called *completion* [KB70]; it works by adding (recursively) new rules whenever a critical pair is found. Usually, completion is associated with a well founded order which tells how to orient rules properly in order to insure the termination of the system. A terminating and locally confluent system is said *convergent.* Newman lemma [New42] tells us that convergent systems are confluent. $\lambda\upsilon$ is not

Syntax. $\Lambda\upsilon$ is the set of terms inductively defined by the following BNF:

$$\begin{aligned}
a, b &::= a\, b \;\mid\; \lambda a \;\mid\; \underline{n} \;\mid\; a[s] && \text{(Terms)}\\
s, t &::= a/ \;\mid\; \Uparrow(s) \;\mid\; \uparrow && \text{(Substitutions)}\\
n &\in \mathbb{N} && \text{(Naturals)}
\end{aligned}$$

Rules. $\lambda\upsilon$ reduction is defined as the reflexive, transitive, contextual, and substitution closure of the following rewriting rules:

$$\begin{aligned}
(\lambda a)\, b &\to a[b/] && \text{(B-}\upsilon\text{)}\\
(a\, b)[s] &\to a[s]\, b[s] && \text{(App-}\upsilon\text{)}\\
(\lambda a)[s] &\to \lambda a[\Uparrow(s)] && \text{(Lambda-}\upsilon\text{)}\\
\underline{0}[a/] &\to a && \text{(FVar-}\upsilon\text{)}\\
\underline{n+1}[a/] &\to \underline{n} && \text{(RVar-}\upsilon\text{)}\\
\underline{0}[\Uparrow(s)] &\to \underline{0} && \text{(FVarlift-}\upsilon\text{)}\\
\underline{n+1}[\Uparrow(s)] &\to \underline{n}[s][\uparrow] && \text{(RVarlift-}\upsilon\text{)}\\
\underline{n}[\uparrow] &\to \underline{n+1} && \text{(VarShift-}\upsilon\text{)}
\end{aligned}$$

Figure 1: The calculus $\lambda\upsilon$

$$\begin{tikzcd}
& & ((\lambda a)\, b)[s] \arrow[d, "(\text{App-}\upsilon)"] \arrow[ddddll, bend right, "(\text{B-}\upsilon)"'] \\
& & (\lambda a)[s]\, b[s] \arrow[d, "(\text{Lambda-}\upsilon)"] \\
& & (\lambda a[\Uparrow(s)])\, b[s] \arrow[d, "(\text{B-}\upsilon)"] \\
& & a[\Uparrow(s)]\big[b[s]/\big] \\
a[b/][s] \arrow[r, dash, dotted] & \textbf{?} \arrow[ur, dash, dotted] &
\end{tikzcd}$$

Figure 2: Non-joinable critical pair of the calculus $\lambda\upsilon$.

terminating (strongly normalising), since it contains the λ-calculus. However, υ itself, which is strongly normalising, is convergent, since it has no critical pair.

Our goal is to obtain a confluent λ-calculus of explicit substitution. For this, we start with $\lambda\upsilon$ and we use an ad-hoc procedure to join the critical pairs of $\lambda\upsilon$, orienting the new rules according to an intuitive order, with no proof of well foundedness. We cross fingers with the hope we are not adding other sources of non-termination as the ones inherent to the λ-calculus.

To join the critical pair of Figure 2, we can add the following rule:

$$a[b/][s] \to a[\Uparrow(s)]\,[b[s]/]$$

(There are many reasons why we do not orient this rule the other way, which we do not develop in this note.) This rule introduces more critical pairs, which can be generated by a completion (which diverges in this case). We are rapidly tempted to add a more general – conditional – rule of the form

$$a[\Uparrow^n(b/)][\Uparrow^{n+m}(s)] \to a[\Uparrow^{n+m+1}(s)][\Uparrow^n(b[\Uparrow^m(s)]/)]$$

where $\Uparrow^{n+1}(s)$ intuitively denotes $\Uparrow\big(\Uparrow^n(s)\big)$ and $\Uparrow^0(s)$ denotes s. We call this rule (P) for *permutation*. The way we write superscripts on lifts *i.e.*, as sums, is a *meta*-notation to avoid writing side conditions. These conditions can be included as part of a sophisticated matching algorithm. For instance, in the left hand side of (P), the condition is that the superscript of the first lift is less or equal than the superscript of the second lift. It is the only case in which one can instantiate n and m, since a natural p is greater or equal than another n if and only if there exists a natural m such that p equals $n+m$. The process leads us also to consider a slightly different way of encoding variables. Instead of encoding them with usual natural numbers, we use the substitution $\uparrow$ as a constructor applied to the variable $\underline{0}$, hence acting like the successor constructor, similarly to the encoding of variables in $\lambda\sigma$ [ACCL91] and others. Hence, the index $\underline{n}$ of $\lambda\upsilon$ will be encoded in the new calculus by the expression

$$\underline{0}\underbrace{[\Uparrow^0(\uparrow)]\ldots[\Uparrow^0(\uparrow)]}_{n \text{ times}}.$$

The process leads us to the locally confluent system of Figure 3, which we call $\lambda\upsilon c$. Our encoding of variables allows us to write more general rules than the ones acting on indexes in $\lambda\upsilon$. Compare for instance (RLift-υc) with (RVarLift-υ), and (R-υc) with (RVar-υ).

Yet, it is not clear that this process preserves the strong normalisation of υ. In other words, it has to be proven that υc is strongly normalising. We show in the next section that this is hopeless.

3 The kid game of permutations

For the sake of termination much care should be done to rules (P-υc) and (RLift-υc), because they permute substitutions. One natural preliminary step in proving strong normalisation, or on the contrary in finding a counter example to strong normalisation, is to address first the strong normalisation of this small subsystem of two rules. In order to draw our attention on an abstraction of this system, we describe it as a kid game.

Syntax. $\Lambda\upsilon c$ is the set of terms inductively defined by the following BNF:

$$a, b ::= \lambda a \mid a\, b \mid \underline{0} \mid a[\Uparrow^n(s)] \qquad \text{(Terms)}$$
$$s, t ::= a/ \mid \uparrow \qquad \text{(Substitutions)}$$
$$n \in \mathbb{N} \qquad \text{(Naturals)}$$

Rules. $\lambda\upsilon c$ reduction is defined as the reflexive, transitive, contextual, and substitution closure of the following rewriting rules:

$$(\lambda a)\, b \to a[\Uparrow^0(b/)] \qquad \text{(B-}\upsilon c\text{)}$$
$$(a\, b)[\Uparrow^n(s)] \to a[\Uparrow^n(s)]\, b[\Uparrow^n(s)] \qquad \text{(App-}\upsilon c\text{)}$$
$$(\lambda a)[\Uparrow^n(s)] \to \lambda a[\Uparrow^{n+1}(s)] \qquad \text{(Lambda-}\upsilon c\text{)}$$
$$\underline{0}[\Uparrow^0(a/)] \to a \qquad \text{(FVar-}\upsilon c\text{)}$$
$$a[\Uparrow^n(\uparrow)][\Uparrow^n(b/)] \to a \qquad \text{(R-}\upsilon c\text{)}$$
$$\underline{0}[\Uparrow^{n+1}(s)] \to \underline{0} \qquad \text{(FVarLift-}\upsilon c\text{)}$$
$$a[\Uparrow^n(\uparrow)][\Uparrow^{n+m+1}(s)] \to a[\Uparrow^{n+m}(s)][\Uparrow^n(\uparrow)] \qquad \text{(RLift-}\upsilon c\text{)}$$
$$a[\Uparrow^n(b/)][\Uparrow^{n+m}(s)] \to a[\Uparrow^{n+m+1}(s)][\Uparrow^n(b[\Uparrow^m(s)]/)] \qquad \text{(P-}\upsilon c\text{)}$$

Figure 3: The calculus $\lambda\upsilon c$

The game of permutations Let white and black be two teams of any numbers of kids. Each player of each team has an arbitrary score, any natural number. All the players are aligned side to side. Then the game processes as follows, where each player can be involved in at most one move at a time:

White move. A player of the white team with a score n can exchange its position with his (her) left hand neighbour (of any team) with a score m provided $m > n$. Then, m is decremented.

Black move. A player of the black team with a score n can exchange its position with his (her) left hand side neighbour (of any team) with a score m provided $m \geq n$. Then, m is incremented.

In our game, the white team abstracts out $\uparrow$ substitutions, so that $\Uparrow^n(\uparrow)$ is interpreted as a white player of score n, while the black team abstracts out $/$ substitutions, so that for any a, $\Uparrow^n(a/)$ is interpreted as a black player of score n. Hence, a white move abstracts out a (P-υc) step, while a black move abstracts out a (RLift-υc) step. Hence, the abstraction is only concerned with what happens on a string $a[s_1]\ldots[s_n]$ of substitutions, but not in proper subterms $s_1, \ldots, s_n$ of this string. The problem is then the following:

Problem 1 *Is it possible to have configurations with which the game of permutations can proceed forever?*

Answer 1 *Unexpectedly to us, the answer is* yes. *Let* black *and* white *have two players each, everyone with score* 0*. Figure 4 shows a configuration which in four moves gets into a similar configuration.*

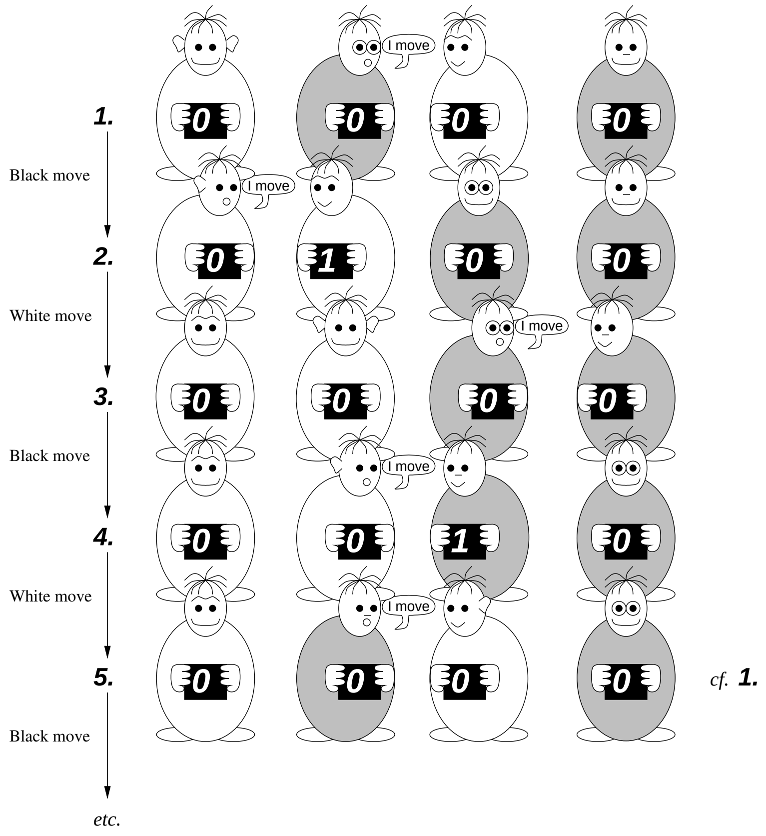

Figure 4: A configuration of the game of permutations leading to an infinite play.

This leads to a direct answer to the problem of strong normalisation of υc.

Corollary 1 *Let a_0, b_0, c be any terms in $\Lambda\upsilon c$, and*

$$a_{n+1} = b_n \qquad b_{n+1} = a_n[\Uparrow^0(\uparrow)][\Uparrow^0(b_n/)]$$

then

$$c[\Uparrow^0(\uparrow)][\Uparrow^0(a_n/)][\Uparrow^0(\uparrow)][\Uparrow^0(b_n/)] \to^+ c[\Uparrow^0(\uparrow)][\Uparrow^0(a_{n+1}/)][\Uparrow^0(\uparrow)][\Uparrow^0(b_{n+1}/)]$$

Proof: It is a four steps reduction using only the two rules (P-υc) and (RLift-υc). The permutable substitutions of the selected redex of each reduction are underlined.

$$\begin{array}{lr}
c[\Uparrow^0(\uparrow)]\underline{[\Uparrow^0(a_n/)][\Uparrow^0(\uparrow)]}[\Uparrow^0(b_n/)] & \\
\quad \to c\underline{[\Uparrow^0(\uparrow)][\Uparrow^1(\uparrow)]}\,[\Uparrow^0(a_n[\Uparrow^0(\uparrow)]/)]\,[\Uparrow^0(b_n/)] & (\text{P-}\upsilon c) \\
\quad \to c[\Uparrow^0(\uparrow)][\Uparrow^0(\uparrow)]\underline{[\Uparrow^0(a_n[\Uparrow^0(\uparrow)]/)]\,[\Uparrow^0(b_n/)]} & (\text{RLift-}\upsilon c) \\
\quad \to c[\Uparrow^0(\uparrow)]\underline{[\Uparrow^0(\uparrow)][\Uparrow^1(b_n/)]}\,[\Uparrow^0(a_n[\Uparrow^0(\uparrow)][\Uparrow^0(b_n/)]/)] & (\text{P-}\upsilon c) \\
\quad \to c[\Uparrow^0(\uparrow)][\Uparrow^0(b_n/)][\Uparrow^0(\uparrow)]\,[\Uparrow^0(a_n[\Uparrow^0(\uparrow)][\Uparrow^0(b_n/)]/)] & (\text{RLift-}\upsilon c) \\
\quad = c[\Uparrow^0(\uparrow)][\Uparrow^0(a_{n+1}/)][\Uparrow^0(\uparrow)][\Uparrow^0(b_{n+1}/)] &
\end{array}$$

□

The following corollary is hence a trivial consequence of the previous one.

Corollary 2 *υc is not strongly normalising.*

4 Non-termination of the calculus τ

Our counter example for υc can be adapted to the calculus τ [Río93], presented in Figure 5. The calculus τ has the same operators as υ, plus the operator $\circ$ which allows to compose substitutions. Thanks to this, the calculus is locally confluent on open terms and, unlike υ and υc, several substitutions can be propagated (under abstractions and applications) in a single step as a single substitution. Indexes in τ are encoded in a similar way as in υc *i.e.*, the index $\underline{n}$ is encoded as $\underline{0}[\underbrace{\uparrow \circ \uparrow \circ \ldots \circ \uparrow}_{n}]$.

The calculus τ has much similarity with υc. In particular, have a look at three of its rules, namely (MapSl-τ) and (ShiftLift i-τ) ($i = 1, 2$), which perform similar permutations as do rules (P-υc) and (RLift-υc). Therefore, we can state the principal result, that τ is not strongly normalising. Note the similarities between Theorem 1 and Corollary 1.

Theorem 1 *Let a_0, b_0 be any terms in $\Lambda\tau$, and*

$$a_{n+1} = b_n \qquad b_{n+1} = a_n[\uparrow \circ\, b_n/]$$

then

$$\uparrow \circ\, (a_n/ \circ (\uparrow \circ\, b_n/)) \to^+ \uparrow \circ\, (a_{n+1}/ \circ (\uparrow \circ\, b_{n+1}/))$$

Proof: It is a three steps reduction, using only three rules of the calculus.

$$\begin{aligned}
\uparrow \circ\, (\underline{a_n/ \circ (\uparrow \circ\, b_n/)}) &\to \underline{\uparrow \circ\, (\Uparrow(\uparrow \circ\, b_n/) \circ a_n[\uparrow \circ\, b_n/]/)} && \text{(MapSl-}\tau\text{)}\\
&\to \underline{(\uparrow \circ\, b_n/) \circ (\uparrow \circ\, a_n[\uparrow \circ\, b_n/]/)} \\
& && \text{(ShiftLift2-}\tau\text{)}\\
&\to \uparrow \circ\, (b_n/ \circ (\uparrow \circ\, a_n[\uparrow \circ\, b_n/]/)) && \text{(AssEnv-}\tau\text{)}\\
&= \uparrow \circ\, (a_{n+1}/ \circ (\uparrow \circ\, b_{n+1}/))
\end{aligned}$$

□

Corollary 3 *τ is not strongly normalising*

To prove that $\lambda\tau$ does not have PSN, we can now show that a pure simply typed λ-term translates into a term in $\Lambda\tau$ which may reduce to a term matching the pattern of the theorem above.

Corollary 4 *$\lambda\tau$ does not preserve strong normalisation.*

Syntax. $\Lambda\tau$ is the set of terms defined inductively by the following BNF:

$$a, b ::= \lambda a \mid a\, b \mid \underline{0} \mid a[s] \qquad \text{(Terms)}$$
$$s, t ::= a/ \mid s \circ t \mid \Uparrow(s) \mid \uparrow \mid \mathrm{id} \qquad \text{(Substitutions)}$$

Rules. $\lambda\tau$ is defined as the reflexive, transitive, contextual, and substitution closure of the following rewriting rules:

$$\begin{array}{rcll}
(\lambda a)\, b & \to & a[b/] & \text{(B-}\tau\text{)}\\
(a\, b)[s] & \to & a[s]\, b[s] & \text{(App-}\tau\text{)}\\
(\lambda a)[s] & \to & \lambda a[\Uparrow(s)] & \text{(Lambda-}\tau\text{)}\\
a[s][t] & \to & a[s \circ t] & \text{(Clos-}\tau\text{)}\\
(s \circ t) \circ u & \to & s \circ (t \circ u) & \text{(AssEnv-}\tau\text{)}\\
a/ \circ s & \to & \Uparrow(s) \circ a[s]/ & \text{(MapSl-}\tau\text{)}\\
\underline{0}[a/] & \to & a & \text{(FVar-}\tau\text{)}\\
\underline{0}[\Uparrow(s)] & \to & \underline{0} & \text{(FVarLift1-}\tau\text{)}\\
\underline{0}[\Uparrow(s) \circ t] & \to & \underline{0}[t] & \text{(FVarLift2-}\tau\text{)}\\
\uparrow \circ\, a/ & \to & \mathrm{id} & \text{(Shift-}\tau\text{)}\\
\uparrow \circ \Uparrow(s) & \to & s \circ \uparrow & \text{(ShiftLift1-}\tau\text{)}\\
\uparrow \circ (\Uparrow(s) \circ t) & \to & s \circ (\uparrow \circ\, t) & \text{(ShiftLift2-}\tau\text{)}\\
\Uparrow(s) \circ \Uparrow(t) & \to & \Uparrow(s \circ t) & \text{(Lift1-}\tau\text{)}\\
\Uparrow(s) \circ (\Uparrow(t) \circ u) & \to & \Uparrow(s \circ t) \circ u & \text{(Lift2-}\tau\text{)}\\
\mathrm{id} \circ s & \to & s & \text{(IdL-}\tau\text{)}\\
s \circ \mathrm{id} & \to & s & \text{(IdR-}\tau\text{)}\\
\Uparrow(\mathrm{id}) & \to & \mathrm{id} & \text{(LiftId-}\tau\text{)}\\
a[\mathrm{id}] & \to & a & \text{(Id-}\tau\text{)}
\end{array}$$

Figure 5: The calculus $\lambda\tau$.

Proof: Consider the closed λ-term $\lambda x.((\lambda y.\lambda z.x)\ x\ x)$. It is simply typable with type $\alpha \to \alpha$. It can be translated into a term in $\Lambda\tau$ as $\lambda\big((\lambda\lambda\underline{0}[\uparrow \circ \uparrow])\ \underline{0}\ \underline{0}\big)$. This one can be reduced as follows:

$$
\begin{array}{rll}
\lambda\big((\lambda\lambda\underline{0}[\uparrow \circ \uparrow])\ \underline{0}\ \underline{0}\big) & \to \lambda\big((\lambda\underline{0}[\uparrow \circ \uparrow])[\underline{0}/]\ \underline{0}\big) & (\text{B-}\tau)\\
& \to \lambda\big((\lambda\underline{0}[\uparrow \circ \uparrow][\Uparrow(\underline{0}/)])\ \underline{0}\big) & (\text{Lambda-}\tau)\\
& \to \lambda(\underline{0}[\uparrow \circ \uparrow][\Uparrow(\underline{0}/)][\underline{0}/]) & (\text{B-}\tau)\\
& \to \lambda(\underline{0}[\uparrow \circ \uparrow][\Uparrow(\underline{0}/) \circ \underline{0}/]) & (\text{Clos-}\tau)\\
& \to \lambda(\underline{0}[(\uparrow \circ \uparrow) \circ (\Uparrow(\underline{0}/) \circ \underline{0}/)]) & (\text{Clos-}\tau)\\
& \to \lambda(\underline{0}[\uparrow \circ (\uparrow \circ (\Uparrow(\underline{0}/) \circ \underline{0}/))]) & (\text{AssEnv-}\tau)\\
& \to \lambda(\underline{0}[\uparrow \circ (\underline{0}/ \circ (\uparrow \circ \underline{0}/))]) & (\text{ShiftLift2-}\tau)
\end{array}
$$

This last term may reduce infinitely, as shows Theorem 1 (take $a_0 = b_0 = \underline{0}$). Hence a strongly normalisable λ-term has an infinite reduction in $\lambda\tau$ *i.e.*, $\lambda\tau$ has not PSN.□

5 Conclusion

$\lambda\tau$ does not preserve strong normalisation of the λ-calculus since τ itself is not even strongly normalizing. We have found a simple counter-example of the termination of τ, thanks to an abstraction of another, similar calculus, namely υc obtained by a "pseudo" completion of $\lambda\upsilon$.

The problem of finding a calculus with 1SIM, MC, and PSN, has now been solved by David and Guillaume [DG99, Gui99] who proposed a calculus called λ_ℓ. The idea of this calculus is to block a certain kind of substitutions in terms, so that they can not interact with others. Recently, Ritter [Rit99] has defined a criterion for calculi which satisfy PSN, and deduced a calculus from $\lambda\sigma_\Uparrow$ which he claims has the expected properties. The criterion is a generalisation of the restriction the first author proposed with Rose in [LR98] (also presented in [Lan98]) for the calculus λxci.

References

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

[BBLRD96] Z. E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, september 1996.

[BR95] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *Proc. of CSN'95*, 1995.

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

[dB78] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. TH-Report 78-WSK-03, Technological University Eindhoven, Netherlands, Department of Mathematics, 1978.

[DG99] R. David and B. Guillaume. The λ_ℓ-calculus. In *Proceedings of the 2nd Int. Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, 1999.

[FKP99] M. C. F. Ferreira, D. Kesner, and L. Puel. Lambda-calculi with explicit substitutions preserving strong normalization. *Applicable Algebra in Engineering, Communication and Computation*, 9(4):333–371, March 1999.

[Gui98] B. Guillaume. The λse calculus does not preserve strong normalisation. submitted, 1998.

[Gui99] B. Guillaume. *Un calcul de substitution avec étiquettes.* PhD thesis, Université de Savoie, 1999.

[KB70] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[KR95] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proc. of PLILP'95*, 1995.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4), 1997.

[Lan98] F. Lang. *Modèles de la β-réduction pour les implantations.* PhD thesis, École Normale Supérieure de Lyon, 1998.

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

[LR98] F. Lang and K. H. Rose. Two equivalent calculi of explicit substitution with confluence on meta-terms and preservation of strong normalisation (one with names and one first order). In *Proceedings of the 1st Int. Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, 1998.

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

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In *Proc. of TLCA'95*, 1995.

[Muñ96] César Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *Proceedings, 11*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 440–447, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[New42] M. H. A. Newman. On theories with a combinatorial definition of equivalence. In *Annals of Math*, volume 43, pages 223–243, 1942.

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

[Rit99] E. Ritter. Characterising explicit substitutions which preserve termination. In *Proceedings of the Int. Conf. on Typed Lambda Calculus and Applications*, 1999.