λ-calculi with explicit substitutions and composition which preserve β-strong normalization (Extended Abstract)

Maria C. F. Ferreira[1] and Delia Kesner[2] and Laurence Puel[2]

[1] Dep. de Informática, Fac. de Ciências e Tecnologia, Univ. Nova de Lisboa, Quinta da Torre, 2825 Monte de Caparica, Portugal, cf@fct.unl.pt.

[2] CNRS & Lab. de Rech. en Informatique, Bat 490, Univ. de Paris-Sud, 91405 Orsay Cedex, France, {kesner,puel}@lri.fr.

Abstract. We study preservation of β-strong normalization by λ_d and λ_{dn}, two confluent λ-calculi with explicit substitutions defined in [10]; the particularity of these calculi is that both have a composition operator for substitutions. We develop an abstract simulation technique allowing to reduce preservation of β-strong normalization of one calculus to that of another one, and apply said technique to reduce preservation of β-strong normalization of λ_d and λ_{dn} to that of λ_f, another calculus having no composition operator. Then, preservation of β-strong normalization of λ_f is shown using the same technique as in [2]. As a consequence, λ_d and λ_{dn} become the first λ-calculi with explicit substitutions having composition and preserving β-strong normalization. We also apply our technique to reduce preservation of β-strong normalization of the calculus λ_v in [14] to that of λ_f.

1 Introduction

The λ-calculus is a convenient framework to study functional languages, with evaluation modeled by the β-reduction rule $(\lambda x.M)N \longrightarrow_\beta M\{x \leftarrow N\}$. The main mechanism used to perform β-reduction is *substitution*, the operation between brackets, which consists of the replacement of formal parameters by actual arguments. This operation is treated as an atomic step in the λ-calculus, and is defined in a meta level by functions external to the language. But substitutions need to be treated explicitly when implementing functional languages, and a mechanism to deal with them becomes necessary. In λ-calculi with explicit substitutions, this is achieved by letting the substitution operation be handled by symbols and reduction rules belonging to the calculus. Most calculi with explicit substitutions also use natural numbers to represent variables (de Bruijn's indices), avoiding in this way the systematic renaming of bound variables (α-conversion) which is necessary in classical λ-calculus to guarantee the correctness of substitution.

In the following, let R-SN stand for R-strong normalization, i. e., strong normalization of the calculus or rule R, and let K stand for λ_K minus the

$Beta$ rule, for any arbitrary calculus λ_K.

The λ_σ-calculus was introduced in [1] as a bridge between the classical λ-calculus and concrete implementations of functional programming languages. Substitutions in λ_σ can be combined by a composition operator. Strong normalization of the σ-calculus is proved in [4], but λ_σ does not preserve β-SN [16]. The λ_σ-calculus is confluent on closed terms but is no longer confluent on open terms. To overcome this, Hardin and Lévy introduced the $\lambda_{\sigma_\Uparrow}$-calculus [9] which allows to recover the confluence property on open terms [3]. However, $\lambda_{\sigma_\Uparrow}$, as λ_σ, does not preserve β-SN either [15].

The λ_υ-calculus [14] also implements the λ-calculus and it has as few forms of substitutions as possible, eliminating the possibility of composing substitutions; λ_υ preserves β-SN [2] but it is not confluent on open terms. Another calculus with explicit substitutions, the λ_s-calculus [12], is inspired by the λ-calculus à la de Bruijn as it explicitly handles the meta-operators used to define substitutions in the de Bruijn's calculus; thus, there are no composition operators for substitutions. λ_s preserves β-SN, however it is not confluent on open terms; an extension of it exists [11] which is confluent on open terms, but for which the preservation of β-SN is still open.

Recently, another calculus with explicit substitutions was proposed in [17]; this calculus is confluent on open terms and preserves β-SN, however, the calculus does not only lack a composition operator for substitutions, but it implements a particular strategy of the λ-calculus: the distribution of substitutions to applications is only possible if the head term is a variable. Thus, there are some β-reduction sequences in classical λ-calculus which can not be simulated in the calculus.

Apparently, there is a choice between confluence on open terms, composition and preservation of β-SN, which is an interesting challenge that we partially answer in this paper: composition is possible in calculi preserving β-SN and being confluent on closed terms. We study λ_d and λ_{dn}, two different calculi with explicit substitutions initially defined in [10]; these calculi, do have composition, are confluent on closed terms[3] and preserve β-SN. The λ_d-calculus is inspired by the λ_τ-calculus from [18], which is in turn inspired by [7], the difference being that the $MapEnv$ rule of λ_τ, namely $\#(a) \circ s \longrightarrow\ \Uparrow(s) \circ \#(a[s])$ is replaced here by the two following rules: $\underline{1}[\#(a) \circ s] \longrightarrow\ a[s]$ and $\uparrow \circ (\#(a) \circ s) \longrightarrow\ s$. This may seem irrelevant, but it is essential to prove confluence and strong normalization of the d-calculus, in contrast with the case of the τ-calculus, for which confluence remains still as a conjecture, and for which strong normalization does not hold, as was

[3] To see why the calculi are not confluent on open terms, consider the term $((\lambda X)a)[s]$ that reduces to $X[\#(a) \circ s]$ and $X[\Uparrow(s) \circ \#(a[s])]$, which cannot be joined.

shown by Pierre Lescanne [13]. However, the rule $MapEnv$ (or a similar one) is essential if one wants to recover the confluence property on open terms, and is at the same time, as explained in [16], the cause of non-termination in many λ-calculi with explicit substitutions. Indeed, analysing [16], it seems that the main difference between our calculi and other calculi that do not preserve β-SN stems from the (in)existence of a rule of the form of $MapEnv$ in λ_τ, or Map in λ_σ.

We show that preservation of β-SN of λ_d and λ_{dn} reduces to that of another calculus, called λ_f. The λ_f-calculus, which can be seen as the λ_υ-calculus of [14] without variables of the form $\underline{n+1}$, arose from transformations of the original calculi λ_d and λ_{dn}. The transformations applied to both calculi work at two levels. First substitution composition is eliminated (and for λ_{dn}, the representation of variables is standardized), giving rise to another intermediate calculus, λ_e. Then, we eliminate the identity substitution from λ_e obtaining λ_f. Those transformations are defined by complete rewrite systems. Reduction sequences in λ_d (resp. λ_{dn}) are translated, via a simulation technique, to reduction sequences in λ_f, in such a way that λ_f-SN is preserved. Consequently, if λ_f preserves β-SN, so do λ_d and λ_{dn} (and the intermediate calculus λ_e). It remains then to see that λ_f does itself preserve β-SN. This is done applying the same technique used in [2]. The simulation technique is an interesting and useful tool; we also apply it to show that preservation of β-SN of λ_υ reduces to that of λ_f.

It would be possible to skip λ_f (thus keeping the identity substitution) and prove preservation of β-SN of λ_e using the technique presented in [2]. But we prefer to work directly with λ_f, which is, in some sense, the minimal calculus of substitutions that one can expect to preserve β-SN and to be confluent on closed terms. Also separating the transformations as opposed to combining both in a single transformation, allows us to break down the problem into much simpler ones (for more technical considerations see Sec. 5).

The paper is organized as follows. In Sec. 2 we introduce some notations and definitions and we give the syntax of all the calculi we are going to deal with. In Sec. 3 we give the simulation technique that will be used along the paper to show that our different calculi preserve β-SN. Sec. 4 is devoted to the translations used by our simulation technique: we define a translation from λ_d to λ_e and another one from λ_e to λ_f. In Sec. 5 we apply the simulation technique introduced in Sec. 3 to show that λ_d (resp. λ_e) preserves λ_e (resp. λ_f) strong normalization. In Sec. 6, a translation from λ_{dn} to λ_e is given; the translation is made in such way that makes it possible to show that λ_{dn} preserves λ_e-SN. In Sec. 7 we show that λ_f preserves β-SN, using a technique from [2]. This result is essential, since it allows us to conclude that λ_d, λ_{dn}, and λ_e also preserve β-SN (Sec. 8). In Sec. 9 we discuss some further topics, including how to reduce preservation of β-SN of λ_υ to that of λ_f. Due to the restrictions of space, we were forced to omit most technical proofs, the reader willing to check the results should consult [8].

2 Definitions and Notations

A classical way to avoid α-conversion problems in λ-calculus, is to use the de Bruijn's notation [5, 6] for λ-terms, where names of variables are replaced by natural numbers. Hence, the set of λ-terms (also called *pure terms*) is defined by the following grammar:

$$Naturals\ n ::= 1 \mid n+1 \qquad\qquad Pure\ Terms\ a ::= \underline{n} \mid aa \mid \lambda a$$

In λ-calculi with explicit substitutions, objects denoting substitutions are incorporated to the language explicitly, as well as manipulation of substitutions, which is incorporated via a set of rewrite rules. This syntactic presentation is defined, for the calculi we are going to present, as follows:

Definition 1 Substitution Signatures. Let us consider two distinguished symbols $\mathcal{T}$ (for terms) and $\mathcal{S}$ (for substitutions). A *substitution declaration* is a (possibly empty) word over the alphabet $\{\mathcal{T}, \mathcal{S}\}$. A *substitution signature* is defined to be a signature Σ (a set of symbols) such that every symbol in Σ is equipped with an arity n and a declaration of length n. We usually write $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$ if ξ has arity n and declaration $\sigma_1 \ldots \sigma_n$.

Definition 2 Substitution Language. The set of objects over a substitution signature Σ is defined to be the union of objects of sort $\mathcal{T}$ and $\mathcal{S}$. They are constructed in the following way:

- If n is a natural number, then $\underline{n}$, called a *variable*, is of sort $\mathcal{T}$.
- If a and b are of sort $\mathcal{T}$, then $(a\ b)$, called an *application*, is of sort $\mathcal{T}$.
- If a is of sort $\mathcal{T}$, then $\lambda(a)$, called an *abstraction*, is of sort $\mathcal{T}$.
- If a is of sort $\mathcal{T}$, s is of sort $\mathcal{S}$, then $a[s]$, called a *closure*, is of sort $\mathcal{T}$.
- If every f_i is of sort σ_i, and $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$, then $\xi(f_1, \ldots, f_n)$, called a *substitution*, is of sort $\mathcal{S}$.

For each substitution language W, the set of objects of sort $\mathcal{T}$ is denoted by $\mathcal{T}_W$, and called *the set of terms of* W. Similarly, $\mathcal{S}_W$, *the set of substitutions of* W, denotes the set of objects of sort $\mathcal{S}$. $\mathcal{O}_W$ stands for $\mathcal{T}_W \cup \mathcal{S}_W$.

We omit parenthesis when association is clear from the context. If W is a substitution language and s is a substitution of W, $[s]^n$ denotes the concatenation of substitutions $\underbrace{[s]\ldots[s]}_{n \quad times}$, assuming that $[s]^0$ denotes the empty word. We also use the following notation for multiple applications of a unary substitution operator: $\xi^0(s) = s$ and $\xi^{n+1}(s) = \xi(\xi^n(s))$.

We write λ_W for the set of rewriting rules of the λ-calculus W, and simply W for the set $\lambda_W - Beta$. The notations $\longrightarrow_{\mathcal{R}}$ and $=_{\mathcal{R}}$ are used to denote resp., $\mathcal{R}$-reduction and $\mathcal{R}$-equality, where $\mathcal{R}$ can be either a set of rewrite rules or a single rule. We abbreviate $\mathcal{R}$-strong normalization to $\mathcal{R}$-SN, and $\mathcal{R}$-strongly normalizing to $\mathcal{R}$-sn. Finally we note that both reduction rules and reductions we are interested in, apply to objects of the appropriate sort, the ones constructed by our grammars.

We now introduce the rewriting rules of the calculi λ_d, λ_{dn}, λ_e and λ_f. The signature associated to the substitution language of the λ_d-calculus is given by the set $\{\#, \Uparrow, \circ, \uparrow, id\}$, where $\# : \langle 1, \mathcal{T}\rangle$, $\Uparrow: \langle 1, \mathcal{S}\rangle$, $\circ : \langle 2, \mathcal{SS}\rangle$, $\uparrow: \langle 0, \epsilon\rangle$ and $id : \langle 0, \epsilon\rangle$; note that these declarations apply to other calculi as well. The set of reduction rules of the λ_d-calculus is given in table 1; these rules can be decomposed into two disjoint sets of rewriting rules:

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(Clos)$	$(a[s])[t]$	$\longrightarrow$	$a[s \circ t]$
$(FVar1)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVar2)$	$\underline{1}[\#(a) \circ s]$	$\longrightarrow$	$a[s]$
$(FVarLift1)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(FVarLift2)$	$\underline{1}[\Uparrow(s) \circ t]$	$\longrightarrow$	$\underline{1}[t]$
(Ass)	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow$	$s_1 \circ (s_2 \circ s_3)$
$(Shift1)$	$\uparrow \circ \#(a)$	$\longrightarrow$	id
$(Shift2)$	$\uparrow \circ (\#(a) \circ s)$	$\longrightarrow$	s
$(ShiftLift1)$	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
$(ShiftLift2)$	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
$(Lift1)$	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
$(Lift2)$	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a[id]$	$\longrightarrow$	a

Table 1. Reduction rules of the λ_d-calculus.

$$
\begin{array}{lll}
(VarShift1) & \underline{n}[\uparrow] & \longrightarrow \underline{n+1} \\
(VarShift2) & \underline{n}[\uparrow \circ s] & \longrightarrow \underline{n+1}[s] \\
(RVar1) & \underline{n+1}[\#(a)] & \longrightarrow \underline{n} \\
(RVar2) & \underline{n+1}[\#(a) \circ s] & \longrightarrow \underline{n}[s] \\
(RVarLift1) & \underline{n+1}[\Uparrow (s)] & \longrightarrow \underline{n}[s \circ \uparrow] \\
(RVarLift2) & \underline{n+1}[\Uparrow (s) \circ t] & \longrightarrow \underline{n}[s \circ (\uparrow \circ t)]
\end{array}
$$

Table 2. Additional reduction rules for the λ_{dn}-calculus (see also table 1).

$$
\begin{array}{lll}
(Beta) & (\lambda a)b & \longrightarrow a[\#(b)] \\
(App) & (a\ b)[s] & \longrightarrow (a[s]\ b[s]) \\
(Lambda) & (\lambda a)[s] & \longrightarrow \lambda(a[\Uparrow (s)]) \\
(FVar) & \underline{1}[\#(a)] & \longrightarrow a \\
(FVarLift) & \underline{1}[\Uparrow (s)] & \longrightarrow \underline{1} \\
(Shift) & a[\Uparrow^i (\uparrow)][\Uparrow^i (\#(b))] & \longrightarrow a[\Uparrow^i (id)] \\
(ShiftLift) & a[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)] & \longrightarrow a[\Uparrow^i (s)][\Uparrow^i (\uparrow)] \\
(LiftId) & a[\Uparrow^{i+1} (id)] & \longrightarrow a[\Uparrow^i (id)] \\
(Id) & a[id] & \longrightarrow a
\end{array}
$$

Table 3. Reduction rules of the λ_e-calculus

- system A, with rules $Beta$, App, $Lambda$, $FVarLift1$, $FVarLift2$, $FVar1$, $FVar2$, $Shift1$, $Shift2$, $ShiftLift1$, $ShiftLift2$, IdL, IdR, $LiftId$ and Id;
- system B, which contains rules $Clos$, Ass, $Lift1$ and $Lift2$.

The signature associated to the λ_{dn}-calculus is the same as before. Reduction rules of the λ_{dn}-calculus are those given for the λ_d-calculus in table 1 plus some additional rules for handling constants of the form $\underline{n}$; the additional rules are given in table 2. The set of rules of the λ_{dn}-calculus can be decomposed into two disjoint sets of rewriting rules:

- system A_n, with rules $Beta$, App, $Lambda$, $FVarLift1$, $FVarLift2$, $FVar1$, $FVar2$, $Shift1$, $Shift2$, $ShiftLift1$, $ShiftLift2$, IdL, IdR, $LiftId$, Id, $RVar1$, $RVar2$, $RVarLift1$ and $RVarLift2$;
- system B_n, which contains rules $Clos$, Ass, $Lift1$, $Lift2$, $VarShift1$ and $VarShift2$.

In λ_e there is no substitution composition, so the signature associated to the substitution language of this calculus is $\{\#, \Uparrow, \uparrow, id\}$. The reduction rules

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
$(\widehat{App})$	$(a\ b)[\Uparrow^i(\uparrow)]$	$\longrightarrow$	$(a[\Uparrow^i(\uparrow)]\ b[\Uparrow^i(\uparrow)])$
	$(a\ b)[\Uparrow^i(\#(c))]$	$\longrightarrow$	$(a[\Uparrow^i(\#(c))]\ b[\Uparrow^i(\#(c))])$
$(\widehat{Lambda})$	$(\lambda a)[\Uparrow^i(\uparrow)]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1}(\uparrow)])$
	$(\lambda a)[\Uparrow^i(\#(c))]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1}(\#(c))])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(\widehat{FVarLift})$	$\underline{1}[\Uparrow^{i+1}(\uparrow)]$	$\longrightarrow$	$\underline{1}$
	$\underline{1}[\Uparrow^{i+1}(\#(a))]$	$\longrightarrow$	$\underline{1}$
$(Shift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^i(\#(b))]$	$\longrightarrow$	$a[\Uparrow^i(id)]$
$(\widehat{ShiftLift})$	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(\uparrow)]$	$\longrightarrow$	$a[\Uparrow^{i+j}(\uparrow)][\Uparrow^i(\uparrow)]$
	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(\#(b))]$	$\longrightarrow$	$a[\Uparrow^{i+j}(\#(b))][\Uparrow^i(\uparrow)]$

Table 4. Reduction rules of the C-calculus

$(\widetilde{App})$	$(a\ b)[\Uparrow^i(id)]$	$\longrightarrow$	$(a[\Uparrow^i(id)]\ b[\Uparrow^i(id)])$
$(\widetilde{Lambda})$	$(\lambda a)[\Uparrow^i(id)]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1}(id)])$
$(\widetilde{FVarLift})$	$\underline{1}[\Uparrow^{i+1}(id)]$	$\longrightarrow$	$\underline{1}$
$(\widetilde{ShiftLift})$	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(id)]$	$\longrightarrow$	$a[\Uparrow^{i+j}(id)][\Uparrow^i(\uparrow)]$
$(LiftId)$	$a[\Uparrow^{i+1}(id)]$	$\longrightarrow$	$a[\Uparrow^i(id)]$
(Id)	$a[id]$	$\longrightarrow$	a

Table 5. Reduction rules of the D-calculus

of λ_e are given in table 3 (the system is infinite since i ranges over $\mathbb{N}$). The reduction rules of λ_e originate two disjoint infinite sets of rewriting rules C and D, shown in tables 4 and 5, resp. (i and j range over $\mathbb{N}$). Even though λ_e does not coincide, syntactically speaking, with the calculus $C \cup D$, when we consider reductions on objects constructed by our grammar, the reduction relations defined by both calculi are the same, i. e.,

$$\text{If } p, p' \in \mathcal{T}_e, \text{ then } p \longrightarrow_{\lambda_e} p' \text{ if and only if } p \longrightarrow_{C \cup D} p'. \qquad (1)$$

As a consequence, from now on we will indistinctly use λ_e or $C \cup D$.

In λ_f there is no substitution composition, nor identity substitution, so the signature associated to the substitution language of λ_f is the set $\{\#, \Uparrow, \uparrow\}$. The reduction rules of λ_f are given in table 6 (i ranges over $\mathbb{N}$).

A variable $\underline{n+1}$ ($n \geq 1$) is represented by the term $\underline{1}[\underbrace{\uparrow \circ (\uparrow \circ (\ldots \circ \uparrow))}_{n\ times}]$ in λ_d, and the term $\underline{1}\underbrace{[\uparrow]\ldots[\uparrow]}_{n\ times}$ in λ_e and λ_f; in every case we can assume the same set of pure terms, given in Sec. 2.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVarLift)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(Shift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^i(\#(b))]$	$\longrightarrow$	a
$(ShiftLift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)]$	$\longrightarrow$	$a[\Uparrow^i(s)][\Uparrow^i(\uparrow)]$

Table 6. Reduction rules of the λ_f-calculus

3 The Simulation Technique

We present the abstract simulation technique used in the rest of the paper.

Definition 3. Let $A = \langle\mathcal{O}, R\rangle$ and $B = \langle\mathcal{O}', R'\rangle$ be two abstract reduction systems such that $\mathcal{O}' \subseteq \mathcal{O}$. We say that A preserves B-SN (or that R preserves R'-SN), if every $a \in \mathcal{O}'$ which is B-sn (R'-sn), is also A-sn (R-sn).

Theorem 4. *Let $R = \langle\mathcal{O}, R_1 \cup R_2\rangle$ be an abstract reduction system such that R_2 is strongly normalizing and there exists a reduction system $S = \langle\mathcal{O}', R'\rangle$, with $\mathcal{O}' \subseteq \mathcal{O}$, and a translation $\mathcal{T}$ from $\mathcal{O}$ to $\mathcal{O}'$ such that $\mathcal{T}(a) = a$, for any $a \in \mathcal{O}'$, and $a \longrightarrow_{R_1} b$ implies $\mathcal{T}(a) \longrightarrow^+_{R'} \mathcal{T}(b)$; $a \longrightarrow_{R_2} b$ implies $\mathcal{T}(a) = \mathcal{T}(b)$. Then $R_1 \cup R_2$ preserves R'-strong normalization.*[4]

4 The Translations

In this section we present the transformations that allow us to go from λ_d to λ_f via the intermediate calculus λ_e. The first transformation is defined by the complete rewrite system de and allows us to eliminate substitution composition from λ_d. The second transformation is defined by the also complete system ef, and allows us to eliminate the identity substitution from λ_e.

From λ_d to λ_e

The following infinite system will be used to translate λ_d to λ_e:

$$(de) \qquad a[\Uparrow^i(s \circ t)] \longrightarrow a[\Uparrow^i(s)][\Uparrow^i(t)] \quad (i \geq 0)$$

We state some properties enjoyed by the system de.

[4] Note that some conditions can be weakened. Namely instead of requiring that for any $a \in \mathcal{O}'$, $\mathcal{T}(a) = a$, we can require that $\mathcal{T}(\mathcal{O}') \subseteq \mathcal{O}'$ and that $\mathcal{T}$ respects strong normalization, i. e., $\mathcal{T}(a)$ is R'-strongly normalizing, if $a \in \mathcal{O}'$ is.

Lemma 5. *The system defined by the rules de is confluent and strongly normalizing. Furthermore if $a \in \mathcal{T}_d$ then $de(a) \in \mathcal{T}_e$, where $de(a)$ is the normal form of a with respect to de.*

Lemma 5 tells us that the system de defines a function, namely the function that associates to each object its unique normal form wrt de. From now on we use the notation $de(s)$ meaning the normal form of s wrt system de. The previous lemma also tells us that de-normal forms of terms from $\mathcal{T}_d$ do not contain substitution composition.

From λ_e to λ_f

The following infinite system will be used to translate λ_e to λ_f:

$$(ef) \qquad a[\Uparrow^i (id)] \longrightarrow a \quad (i \geq 0)$$

Lemma 6. *The system ef is confluent and strongly normalizing. Furthermore if $a \in \mathcal{T}_e$ then $ef(a) \in \mathcal{T}_f$, where $ef(a)$ is the normal form of a wrt ef.*

Just like the system de, the system ef defines a function that associates to each object in $\mathcal{O}_e$ its normal form wrt ef. It also states that ef-normal forms of terms of $\mathcal{T}_e$, do not contain the substitution id.

5 Simulating the λ_d-calculus

We show that λ_d preserves λ_f-SN. For that, we proceed in two steps, by showing first that λ_d preserves λ_e-SN, and then, that λ_e preserves λ_f-SN.

λ_d preserves λ_e-strong normalization

We consider the partition $A \cup B$ of the λ_d-calculus; B is a sub-calculus of $\sigma_{\Uparrow}$, which is strongly normalizing [9], so we can state:

Proposition 7. *The B-calculus is strongly normalizing.*

Lemma 8. *Let p, p' be objects in $\mathcal{O}_d$ such that $p \longrightarrow_{\lambda_d} p'$.*

1. *If $p \in \mathcal{T}_d$, then $p \longrightarrow_A p'$ implies $de(p') \longrightarrow^+_{\lambda_e} de(p')$, and $p \longrightarrow_B p'$ implies $de(p') = de(p')$.*
2. *$\forall d \in \mathcal{T}_e$ if $d[p] \in \mathcal{T}_d$, then $\forall i \geq 0$:*
   $p \longrightarrow_A p'$ implies $de(d[\Uparrow^i (p)]) \longrightarrow^+_{\lambda_e} de(d[\Uparrow^i (p')])$
   $p \longrightarrow_B p'$ implies $de(d[\Uparrow^i (p)]) = de(d[\Uparrow^i (p')])$

Corollary 9. *Let $a, b \in \mathcal{T}_d$. If $a \longrightarrow_A b$, then $de(a) \longrightarrow^+_{\lambda_e} de(b)$, and if $a \longrightarrow_B b$, then $de(a) = de(b)$.*

Corollary 10. *λ_d preserves λ_e-strong normalization.*

Proof. Since $t \in \mathcal{T}_e$ does not contain occurrences of the symbol $\circ$, then $de(t) = t$. Combining this with proposition 7 and corollary 9 allows us to use theorem 4 (with $\mathcal{O}$ as $\mathcal{T}_d$, R_1 as A, R_2 as B, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as de), to conclude that λ_d preserves λ_e-strong normalization.

λ_e preserves λ_f-strong normalization

We will actually prove that the $(C \cup D)$-calculus preserves λ_f-SN. Due to the fact 1, in Sec. 2, we can then immediately conclude that λ_e preserves λ_f-SN.

Proposition 11. *The D-calculus is strongly normalizing.*

For a proof of the previous result, see [8]. As in lemma 8 one can show:

Lemma 12. *Let p, p' be terms in $\mathcal{T}_e$ such that $p \longrightarrow_{\lambda_e} p'$. Then, $p \longrightarrow_C p'$ implies $ef(p) \longrightarrow^+_{\lambda_f} ef(p')$, and $p \longrightarrow_D p'$ implies $ef(p) = ef(p')$.*

As remarked in Sec. 2, fact 1, whenever p and p' are terms in $\mathcal{T}_e$, then $p \longrightarrow_{\lambda_e} p'$ if and only if $p \longrightarrow_{C \cup D} p'$. As a consequence, we have:

Corollary 13. *The calculus λ_e preserves λ_f-strong normalization.*

Proof. For $t \in \mathcal{T}_f$, $ef(t) = t$, since t does not contain occurrences of the identity substitution. Combining this with proposition 11 and lemma 12 allows us to use theorem 4 (with $\mathcal{O}$ as $\mathcal{T}_e$, R_1 as C, R_2 as D, $\mathcal{O}'$ as $\mathcal{T}_f$, R' as λ_f and $\mathcal{T}$ as ef) to conclude that $C \cup D$ preserves λ_f-SN. Due to the observation above, the result follows.

Some considerations about the transformations

We make some relevant comments on the strategy used. Our aim is to prove β-SN of the λ_d-calculus, so we transform this calculus by eliminating both substitution composition and the identity substitution, and this is done in two steps via the complete rewrite systems de and ef. It is possible to put these systems together to form a system, say $\mathcal{DF}$, that turns out to be also complete. Therefore this system $\mathcal{DF}$ would define a function we could use to translate the calculus λ_d into the calculus λ_f. The problem is that instead of obtaining a simple partition of λ_d like the systems A and B, we obtain a partition that is constituted by two conditional rewrite systems. By separating the transformation, we obtain one more calculus but the solution is cleaner and definitively simpler, since no conditional systems are involved.

6 Simulating the λ_{dn}-calculus

Preservation of β-SN of the λ_{dn}-calculus can also be reduced to preservation of β-SN of the λ_e-calculus (for pure terms). The proof follows closely the proof of the same result for the λ_d-calculus: one defines a translation from the λ_{dn}-calculus to the λ_e-calculus, and proves that λ_{dn} preserves λ_e-SN.

To define the translation, consider the following infinite rewrite system:

$$\begin{array}{lll} (dne) & a[\Uparrow^i(s \circ t)] \longrightarrow & a[\Uparrow^i(s)][\Uparrow^i(t)] \quad (i \geq 0) \\ & \underline{n+1} \qquad\quad \longrightarrow & \underline{1}[\uparrow]^n \quad (n \geq 1) \end{array}$$

The system dne is confluent and strongly normalizing; furthermore

Lemma 14. *If $a \in \mathcal{T}_{dn}$ then $dne(a) \in \mathcal{T}_e$.*

In order to apply the simulation techniques, one needs the following:

Lemma 15. *Let p, p' be objects in $\mathcal{O}_{dn}$ such that $p \longrightarrow_{\lambda_{dn}} p'$. Then*

1. *If p is a term, then* $\begin{array}{l} p \longrightarrow_{A_n} p' \textit{ implies } dne(p) \longrightarrow^+{}_{\lambda_e} dne(p') \\ p \longrightarrow_{B_n} p' \textit{ implies } dne(p) = dne(p') \end{array}$
2. *If p is an object, then for every $d \in \mathcal{T}_e$ such that $d[p] \in \mathcal{T}_{dn}$, and every $i \geq 0$ we have* $\begin{array}{l} p \longrightarrow_{A_n} p' \textit{ implies } dne(d[\Uparrow^i(p)]) \longrightarrow^+{}_{\lambda_e} dne(d[\Uparrow^i(p')]) \\ p \longrightarrow_{B_n} p' \textit{ implies } dne(d[\Uparrow^i(p)]) = dne(d[\Uparrow^i(p')]) \end{array}$

Corollary 16. *Let $a \in \mathcal{T}_{dn}$. If $a \longrightarrow_{A_n} b$, then $dne(a) \longrightarrow^+{}_{\lambda_e} dne(b)$, and if $a \longrightarrow_{B_n} b$, then $dne(a) = dne(b)$.*

In order to see that the λ_{dn}-calculus preserves λ_e-SN, we still need to show that the the B_n-calculus is strongly normalizing. This is stated in the following proposition (for the full proof see [8]).

Proposition 17. *The B_n-calculus is strongly normalizing.*

Corollary 18. *The λ_{dn}-calculus preserves λ_e-strong normalization.*

Proof. If $t \in \mathcal{T}_e$, t does not contain occurrences of the symbol $\circ$ nor any variable $\underline{n+1}$, so $dne(t) = t$. Combining this with proposition 17 and corollary 16 allows us to use theorem 4 (with $\mathcal{O}$ as $\mathcal{T}_{dn}$, R_1 as A_n, R_2 as B_n, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as dne), to conclude that λ_{dn} preserves λ_e-SN.

7 Preservation of β-Strong Normalization of the λ_f-calculus

We show that the λ_f-calculus preserves β-SN for pure terms. The proof technique used is a minimality construction similar to the one used in [2].

Theorem 19. *The f-calculus is strongly normalizing and confluent.*

So the system f defines a function, namely the function that given any object in $\mathcal{O}_f$ associates to it its unique normal form wrt the f-calculus.

Proposition 20. *For every object p in $\mathcal{O}_f$, (1) if p is a term, then $f(p)$ is a pure term; and (2) for every pure term d, if $d[p] \in \mathcal{T}_f$, then for every $i \geq 0$, $f(d[\Uparrow^i (p)])$ is a pure term.*

Corollary 21. *f-normal forms of terms are pure terms.*

Theorem 22 Projection Lemma. *Let a, b be terms in $\mathcal{T}_f$. If $a \longrightarrow_{Beta} b$, then $f(a) \longrightarrow^*_\beta f(b)$.*

For a proof of the previous result, we refer the reader to [8].

A *position* in an object $o \in \mathcal{O}_f$ is a word over $\{1, 2\}$ satisfying:

- $o_{|_\epsilon} = o$ (ϵ is the empty word);
- $o_{|_{p \cdot 1}} = a$, if $o_{|_p}$ is either (ab), λa, $a[s]$, $\#(a)$ or $\Uparrow (a)$;
- $o_{|_{p \cdot 2}} = b$, if $o_{|_p}$ is either (ab) or $a[b]$.

The set of all positions of an object o is denoted by $Pos(o)$. We want to distinguish between *external* and *internal* positions. Intuitively, internal positions are the positions that occur inside [ ] symbols; external positions are the positions occurring outside the outermost [ ] symbol. Given a term $t \in \mathcal{T}_f$, the set of its external, resp. internal, positions is denoted by $Ext(t)$, resp. $Int(t)$. These sets are inductively defined as follows:

- if $t = \underline{1}$, then $Ext(t) = \{\epsilon\}$ and $Int(t) = \emptyset$;
- if $t = (a\ b)$, then $Ext(t) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{2 \cdot p |\ p \in Ext(b)\} \cup \{\epsilon\}$, and $Int(t) = \{1 \cdot p |\ p \in Int(a)\} \cup \{2 \cdot p |\ p \in Int(b)\}$;
- if $t = \lambda a$, then $Ext(t) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{\epsilon\}$, and $Int(t) = \{1 \cdot p |\ p \in Int(a)\}$;
- if $t = a[s]$ then $Ext(t) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{\epsilon\}$, and $Int(t) = \{1 \cdot p |\ p \in Int(a)\} \cup \{2 \cdot p |\ p \in Pos(s)\}$.

It is not difficult to see that a position in a term is either internal or external, i. e., for any term $t \in \mathcal{T}_f$, $Pos(t) = Ext(t) \cup Int(t)$. Furthermore the notion of external/internal can be extended to reduction steps. For $a_1, a_2 \in \mathcal{T}_f$, we say that the reduction is external (resp. internal) and denote it by

$a_1 \stackrel{ext}{\longrightarrow} a_2$ (resp. $a_1 \stackrel{int}{\longrightarrow} a_2$) if $a_1 \longrightarrow a_2$ and the reduction occurs at an external (resp. internal) position in a_1.

Proposition 23. *Let $a, b \in \mathcal{T}_f$. If $a \stackrel{ext}{\longrightarrow}_{Beta} b$ then $f(a) \longrightarrow^+_\beta f(b)$.*

Lemma 24 Commutation Lemma. *Let $a, b \in \mathcal{T}_f$ be such that $f(a)$ is β-sn and $f(a) = f(b)$. If $a \stackrel{int}{\longrightarrow}_{\lambda_f} . \stackrel{ext}{\longrightarrow}_f b$, then $a \stackrel{ext}{\longrightarrow}^+_f . \stackrel{int}{\longrightarrow}_{\lambda_f} b$.*

Lemma 25 Iterative Commutation Lemma. *[2] Let $a_0, \cdots, a_n$ be $n+1$ terms in $\mathcal{T}_f$, such that $f(a_0)$ is β-sn and $f(a_i) = f(a_j)$, for all $0 \leq i, j \leq n$; and $a_i \stackrel{int}{\longrightarrow}^*_{\lambda_f} \cdot \stackrel{ext}{\longrightarrow}_f a_{i+1}$, for all $0 \leq i \leq n-1$. Then $a_0 \stackrel{n,ext}{\longrightarrow}_f \cdot (\stackrel{int}{\longrightarrow}_{\lambda_f} \cup \stackrel{ext}{\longrightarrow}_f)^* a_n$.*

Lemma 26. *Let a be a β-sn pure term. For every infinite λ_f-derivation $a = b_0 \longrightarrow_{\lambda_f} b_1 \longrightarrow_{\lambda_f} b_2 \longrightarrow_{\lambda_f} \ldots$, there exists $N \geq 0$ such that for $i \geq N$ all the reduction steps $b_i \longrightarrow_{\lambda_f} b_{i+1}$ are internal.*

Theorem 27. *If a is a β-sn pure term, then it is λ_f-sn.*

The previous result is the main result of this section and relies heavily on the preceding ones. The proof proceeds by contradiction; we admit there is an infinite λ_f-derivation minimal in some sense and arrive at a smaller derivation. Many details have been left out; the reader should consult [8].

8 λ_d and λ_{dn} preserve β-strong normalization

We present our main result: preservation of β-SN by λ_d and λ_{dn}.

Theorem 28. *If a is a β-sn pure term, then it is λ_e, λ_d and λ_{dn}-sn.*

Proof. Let a be a β-sn pure term. By theorem 27, a is also λ_f-sn and since λ_e preserves λ_f-SN (by corollary 13), we conclude that a is λ_e-sn. Since λ_d preserves λ_e-SN (by corollary 10), we conclude that a is λ_d-sn, and since λ_{dn} preserves λ_e-SN (by corollary 18), we conclude that a is λ_{dn}-sn.

9 Further topics

Preservation of β-SN of the λ_υ-calculus also reduces to preservation of β-SN of the λ_f-calculus. The proof can be done using the same simulation technique explained in sections 3 and 5, where the system used to translate the λ_υ-calculus to the λ_f-calculus is given by the following rewrite system:

$$(vf) \qquad \underline{n+1} \longrightarrow \underline{1}[\uparrow]^n \quad (n \geq 1)$$

Consider the $VarShift$ rule of λ_υ, defined as $\underline{n}[\uparrow] \longrightarrow \underline{n+1}$; taking $\mathcal{O}$ as $\mathcal{T}_\upsilon$, R_1 as $\lambda_\upsilon - VarShift$, R_2 as $VarShift$, $\mathcal{O}'$ as $\mathcal{T}_f$, R' as λ_f and $\mathcal{T}$ as vf in theorem 4, we can state that λ_υ preserves λ_f-SN. and so we can conclude that, as shown in [2], λ_υ preserves β-SN.

We have shown preservation of β-SN of two different λ-calculi with explicit substitutions (λ_d and λ_{dn}) which have composition of substitutions and are confluent on closed terms. For the proof, an abstract simulation technique was developed to reduce β-SN of these calculi to that of another one. We applied this technique to show that, not only preservation of β-SN for λ_d and λ_{dn}, but also for λ_υ, reduces to preservation of β-SN of λ_f, which is a very simple calculus containing no composition and no identity substitution. Thus, λ_f becomes the minimal calculus of explicit substitutions that one can expect to preserve β-SN and to be confluent on closed terms, while λ_d and λ_{dn} become the first λ-calculi with explicit substitutions having composition, being confluent on closed terms and preserving β-SN.

Next challenge concerns the definition of a calculus in the same spirit of λ_d (having composition and preserving β-SN) but being confluent on *open* terms. We are also interested in a *general scheme* for substitution calculi as the one defined in [10] to study preservation of β-SN from a more abstract point of view. Finally, it would be interesting to investigate which are the abstract functional machines that one can encode using λ_d or λ_{dn}.

Acknowledgments This work was partially done when M. Ferreira visited LRI at the University of Paris-Sud, in January 1996. She was partially supported by NWO, the Dutch Organization for Scientific Research, under grant 612-316-041, the University of Utrech, and by a Human Capital and Mobility project (#ERBCHRXCT940495).

References

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

[2] Z.-El-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\nu$, a calculus of explicit substitutions which preserves strong normalisation, 1995. Available from `http://www.loria.fr/ lescanne/publications.html`.

[3] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report 1617, INRIA Rocquencourt, 1992.

[4] P.-L. Curien, T. Hardin, and A. Ríos. Strong normalisation of substitutions. In *MFCS'92*, number 629 in LNCS, pages 209–218, 1992.

[5] N. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Mat.*, 5(35):381–392, 1972.

[6] N. de Bruijn. Lambda-calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, (40):384–356, 1978.

[7] T. Ehrhard. *Une sémantique catégorique des types dépendants. Application au calcul des constructions.* Thèse de doctorat, Université de Paris VII, 1988.

[8] M. C. F. Ferreira, D. Kesner and L. Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization, 1996. Available via ftp at ftp.lri.fr/LRI/articles/kesner/psn.ps.gz.

[9] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[10] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. In *Proc. of the Seventh Int. Conf. RTA*, 1996. To appear.

[11] F. Kamareddine and A. Ríos. The confluence of the λs_e-calculus via a generalized interpretation method. Technical report TR-1996-19, Dep. of Computing Science, University of Glasgow, 1996.

[12] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proc. of the Int. Symposium on Programming Language Implementation and Logic Programming*, number 982 in LNCS. Springer-Verlag, 1995.

[13] P. Lescanne. Personal Communication. 1996.

[14] P. Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *Ann. ACM Symp. POPL*, pages 60–69. ACM, 1994.

[15] P.-A. Mellies. Four typed-lambda calculi with explicit substitutions may not terminate: the first examples, 1994. Draft.

[16] P.-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. of Int. Conf. TLCA*, number 902 in LNCS, April 1995.

[17] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *Proc. of the Symposium LICS*, 1996 To appear.

[18] A. Ríos. *Contribution à l'étude des λ-calculus avec substitutions explicites.* Thèse de doctorat, Université de Paris VII, 1993.

This article was processed using the LaTeX macro package with LLNCS style