Explicit substitution : on the edge of strong normalisation

Citation for published version (APA):

Document status and date:
Published: 01/01/1996

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Explicit Substitution: on the Edge of Strong Normalisation

by

R. Bloo and H. Geuvers

96/10

ISSN 0926-4515

All rights reserved
editors: prof.dr. R.C. Backhouse
prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Report 96/10
Eindhoven, April 1996
Explicit Substitution: on the Edge of Strong Normalisation

Roel Bloo*  
lbloo@win.tue.nl
Herman Geuvers  
herman@win.tue.nl

1 Abstract

We use the Recursive Path Ordering (RPO) technique of semantic labelling to show the Preservation of Strong Normalisation (PSN) property for several calculi of explicit substitution. Preservation of Strong Normalisation states that if a term M is strongly normalizing under ordinary β-reduction (using 'global' substitutions), then it is strongly normalizing if the substitution is made explicit ('local'). There are different ways of making global substitution explicit and PSN is a quite natural and desirable property for the explicit substitution calculus. Our method for proving PSN is very general and applies to several known systems of explicit substitutions: λν of Lescanne et al., λα of Kamareddine and Rios and λx of Rose and Bloo.

Keywords: lambda-calculus, explicit substitution, recursive path order.

2 Introduction

Explicit Substitution was first studied by Abadi, Cardelli, Curien and Lévy in [Abadi et al. 90]. They proposed a calculus λσ of explicit substitutions which can compose substitutions. Mellies has shown that simply typable terms can have infinite reduction paths in λσ ([Mellies 95]). Several people (see [BBLR 95],[Bloo & Rose 95],[Bloo 95],[Kamareddine & Rios 95]) have succeeded in giving calculi of explicit substitutions which have the nice property that every term which is strongly normalising for β-reduction is also strongly normalising in the explicit substitution calculus. We call this property: PSN (Preservation of Strong Normalisation).

In this paper we present a method to prove PSN for explicit substitution calculi based on the recursive path order. Zantema used the recursive path order to show termination of the substitution part of λσ [Zantema 94], but the technique he used doesn't apply to show PSN. We use a stronger technique called semantic labelling [Ferreira & Zantema 94] to show PSN for all explicit substitution calculi known to have the PSN property. We also show why our method doesn't work for λσ. Our technique relies on introducing a first order term rewrite system where function symbols for application and substitution are labeled with natural numbers and where variables are represented by just one constant *. The recursive path order >rpo on this labelled calculus is strongly normalising (or: terminating).

Then we take a look at the explicit substitution calculus λx. The β-reduction is here split up into a reduction step \( \rightarrow_{\beta\text{eta}} \) (contracting the β-redex and creating an explicit substitution) and reduction steps \( \rightarrow_x \) (moving the explicit substitutions through the term to perform the substitution). It is relatively easy (as usual in these calculi) to observe that \( \rightarrow_x \) is strongly normalising and confluent.

Now—and this is a crucial point in the proof of PSN—we take a look at the terms in λx of which the substitution normal form of all of its subterms is β-SN; we call this set \( \lambda x<\infty \). (The substitution normal form of a term M is obtained by evaluating all the explicit substitutions in M, not contracting any β-redexes. That is, we take the \( \rightarrow_{x} \)-normal-form of M.) We then define a translation \( T \) from \( \lambda x<\infty \) into the set of labelled terms. This translation \( T \) is reduction preserving.

*address of both authors: Eindhoven University of Technology, P.O.Box 513, NL-5600 MB Eindhoven.
in the sense that, if \( M \rightarrow_{x} N \), then \( T(M) >_{\text{sto}} T(N) \) or \( T(M) = T(N) \) and if \( M \rightarrow_{\text{Beta}} N \), then \( T(M) >_{\text{sto}} T(N) \). Hence, using the fact that \( \rightarrow_{x} \) is strongly normalizing, we conclude that every \( M \in \lambda x ^{< \infty} \) is strongly normalizing. So, \( \lambda x \) has the PSN property (because every \( \lambda \)-term that is \( \beta \)-strongly-normalizing is an element of \( \lambda x ^{< \infty} \)).

For those more familiar with the RPO technique in the way it has been presented in [Klop 92], we also present, in the final section, a translation from \( \lambda x ^{< \infty} \) to commutative labelled trees. This translation is also reduction preserving in the same sense as just discussed. This gives a slightly different way of obtaining the PSN property for \( \lambda x \).

To show the flexibility of our proof method we use it for different calculi of explicit substitution. We start off with a calculus where we use named variables (different from, e.g. [Abadi et al. 90], where de Bruijn-indices are used). We have chosen to use named variables because this makes the presentation slightly more perspicuous. Moreover, it makes it easier to single out the places where the difficulties arise in the calculus of [Abadi et al. 90]. We also apply our proof method to the calculi \( \lambda \nu \) of Lescanne et al. and \( \lambda s \) of Kamareddine and Rios.

### 3 A calculus for explicit substitutions with named variables

In the standard definition of the untyped lambda calculus, substitution is a meta-operation, usually denoted by \( [x := N] \) or \( [N/x] \), where \( x \) is a variable and \( N \) a term. In the following we use the notation \( [N/x] \) for a (global) substitution of \( N \) for \( x \). For \( M \) and \( N \) terms and \( x, y \) distinct variables, the term \( M[N/x] \) is then defined by structural induction as follows.

\[
\begin{align*}
x[N/x] &:= N, \\
y[N/x] &:= y, \text{ if } y \neq x, \\
(PQ)[N/x] &:= P[N/x]Q[N/x], \\
(\lambda y.P)[N/x] &:= \lambda y'.P[y'/y][N/x], \text{ if } y' \notin \text{ FV}(N) \cup \{x\} \cup (\text{FV}(P) \setminus \{y\}) \\
(\lambda x.P)[N/x] &:= \lambda z.P.
\end{align*}
\]

We assume the notions of free variable (FV) and bound variable (BV) to be known. Furthermore, \( \equiv \) denotes syntactical equality modulo \( \alpha \)-conversion, which is defined as the smallest equivalence relation such that

\[
\begin{align*}
x &\equiv z, \\
P \equiv N \text{ and } Q \equiv M &\Rightarrow PQ \equiv NM, \\
P \equiv Q, y \notin \text{ FV}(Q) \setminus \{x\} &\Rightarrow \lambda z.P \equiv \lambda y.Q[y/x].
\end{align*}
\]

In the definition of substitution, there is a choice for the variable \( y' \). For this definition to make sense, it has to be shown that the specific choice for the variable \( y' \) is irrelevant. But this is a consequence of the definition of \( \equiv \) and the following Lemma.

**Lemma 3.1** If \( P \equiv Q \) and \( M \equiv N \), then \( P[M/x] \equiv Q[N/x] \).

In order to get a calculus \( \lambda x \) of explicit substitutions, two extensions have to be made. The first is extending the terms with substitutions:

**Definition 3.2** The set of terms \( \lambda x \) is defined by the following abstract syntax:

\[
A ::= z \mid AA \mid \lambda z.A \mid A(x := A)
\]

Where \( x \) denotes an arbitrary variable.

Substitution is defined on \( \lambda x \)-terms as for \( \lambda \)-terms but with the extra clauses that

\[
M(y := P)[N/x] \equiv M[y'/y][N/x][y' := P[N/x]] \text{ if } y' \notin \text{ FV}(N) \cup \{x\} \cup (\text{FV}(M) \setminus \{y\})
\]
\[ M(x:=P)[N/x] \equiv M(x:=P[N/x]) \]

\( \alpha \)-equivalence is defined on \( \lambda x \)-terms as for \( \lambda \)-terms but with the extra clause that

\[ M \equiv N, P \equiv Q, y \notin \text{FV}(Q) \setminus \{x\} \Rightarrow P(x:=M) \equiv Q[y/x](y:=N) \]

\( A \in \lambda x \) is called pure if \( A \notin \lambda \), i.e., \( A \) does not contain any substitution \( (x:=B) \).

The second is refining the notion of \( \beta \)-reduction. Remember that the reduction relation \( \rightarrow_{\beta} \)
on pure terms is defined as the contextual closure of

\[ (\lambda x.A)(x:=B) \rightarrow_{\beta} A[B/x] \]

We make the global substitution in \( \rightarrow_{\beta} \) explicit by splitting \( \rightarrow_{\beta} \) into two parts. \( \rightarrow_{\text{Beta}} \) is for the creation of a substitution out of a \( \beta \)-redex; \( \rightarrow_{x} \) is for the proliferation of substitutions through a term to variables and for performing the actual substitution or throwing away the substitute if the substitution turns out to be void.

**Definition 3.3** The reduction relations \( \rightarrow_{\text{Beta}} \) and \( \rightarrow_{x} \) are defined to be the contextual closures modulo \( \alpha \)-conversion of respectively

\[ (\lambda x.A)(x:=B) \rightarrow_{\text{Beta}} A[x:=B] \]

and

\[
\begin{align*}
(AB)(x:=C) & \rightarrow_{x} (A(x:=C))(B(x:=C)) \\
(\lambda y.A)(x:=C) & \rightarrow_{x} \lambda y.A(x:=C) \text{ if } x \notin y \text{ and } y \notin \text{FV}(C) \\
z(x:=C) & \rightarrow_{x} C \\
A(x:=C) & \rightarrow_{x} A \text{ if } x \notin \text{FV}(C)
\end{align*}
\]

The explicit substitution reduction relation \( \rightarrow_{\lambda x} \) is the union of \( \rightarrow_{\text{Beta}} \) and \( \rightarrow_{x} \).

The reduction \( A(x:=C) \rightarrow_{x} A \) if \( x \notin \text{FV}(A) \) is also called garbage collection. Since we consider terms modulo \( \alpha \)-equality, substitutions can always be distributed to variables, hence the rule \( y(x:=C) \rightarrow_{x} y \) if \( x \neq y \) would already be sufficient. The more efficient garbage collection will do no harm however.

The reduction relation \( \rightarrow_{x} \) is called the substitution calculus. It has nice properties:

**Lemma 3.4** \( \rightarrow_{x} \) is strongly normalising, confluent and has unique normalforms.

**Proof:** Strong normalisation is shown by defining a map \( h : \lambda x \rightarrow N \) which decreases on \( x \)-reduction; define

\[
\begin{align*}
h(x) & = 1 \\
h(AB) & = h(A) + h(B) + 1 \\
h(\lambda x.A) & = h(A) + 1 \\
h(A(x:=B)) & = h(A) \cdot (h(B) + 1)
\end{align*}
\]

then by induction on the structure of \( A \): if \( A \rightarrow_{x} B \) then \( h(A) > h(B) \).

To prove confluence, it is now sufficient to show weak confluence which is easy. \( \square \)

**Definition 3.5** We write \( A \in SN_{R} \) if \( A \) is strongly normalizing with respect to the reduction relation \( R \).

We write \( x(A) \) to denote the \( x \)-normalform of \( A \).

For pure terms \( A \), we write \( \beta(A) \) to denote the \( \beta \)-normalform of \( A \), if it exists.

We define \( \beta(A) \) to be the maximal length of a \( \beta \)-reduction path starting with \( x(A) \), if \( x(A) \in SN_{\beta} \).
Note that for $A \in \lambda x, x(A)$ is pure.

We now give some elementary but important properties of $x$ and $\beta$.

**Lemma 3.6 (substitution)** For all terms $A, B$: $x(A(x:=B)) \equiv x(A)[x(B)/x]$.

**Proof:** We prove by induction on the number of symbols in the sequence $A, B_1, \ldots, B_m$ that $x(A(x_1:=B_1) \cdots (x_m:=B_m)) \equiv x(A)[x(B_1)/x_1] \cdots [x(B_m)/x_m]$.

We distinguish cases according to the structure of $A$; we only treat some of them:

- $A \equiv x_i$. Then $x(A(x_1:=B_1) \cdots (x_m:=B_m)) \equiv x(B_i(x_{i+1}:=B_{i+1}) \cdots (x_m:=B_m)) \overset{IH}{=} x(B_i)[x(B_{i+1})/x_{i+1}] \cdots [x(B_m)/x_m] \equiv x(A)[x(B_i)/x_1] \cdots [x(B_m)/x_m]$.

- $A \equiv A_1(y:=A_2)$. Then the number of symbols in the sequence $A, B_1, \ldots, B_m$ is larger than in the sequence $A_1, A_2, B_1, \ldots, B_m$ and the number of symbols in $A$ is bigger than in $A_1, A_2$ so by the induction hypothesis:
  
  $x(A_1(y:=A_2)(x_1:=B_1) \cdots (x_m:=B_m)) \equiv x(A_1)[x(A_2)/y][x(B_1)/x_1] \cdots [x(B_m)/x_m] \equiv x(A_1(y:=A_2))[x(B_1)/x_1] \cdots [x(B_m)/x_m]$.

\[ \square \]

**Lemma 3.7 (projection)** For all terms $A, B$:

1. if $A \rightarrow_{\lambda} B$ then $x(A) \equiv x(B)$
2. if $A \rightarrow_{\beta} B$ then $x(A) \rightarrow_{\beta} x(B)$

**Proof:**

1. is immediate and 2. is by induction on the structure of $A$. Note that if $N \rightarrow_{\beta} N'$ then $M[N/x] \rightarrow_{\beta} M[N'/x]$. We treat some cases:

- $A \equiv (\lambda x. A_1) A_2, B \equiv A_1(x:=A_2)$. Then $x(A) \equiv (\lambda x. A_1) x(A_2) \rightarrow_{\beta} x(A_1)[x(A_2)/x] \overset{3.6}{=} x(A_1[x:=A_2]) \equiv x(B)$.

- $A \equiv A_1(x:=A_2), B \equiv A_1(x:=A_2)$. Then $x(A) \overset{3.6}{=} x(A_1)[x(A_2)/x] \overset{IH}{=} x(A_1)[x(A_2)/x] \overset{3.6}{=} x(B)$.

\[ \square \]

The projection lemma is not strong enough to give us PSN. The problem is that if $A \rightarrow_{\beta} B$ then sometimes $x(A) \equiv x(B)$, as in $x(y=(\lambda z.C)D) \rightarrow_{\beta} x(y=C;z:=D))$. A proof of PSN by analyzing what can happen inside 'void' substitutions such as in this example is given in [Bloo 95] and in [Bloo & Rose 95].

**Lemma 3.8 (soundness)** For all pure terms $A, B$: if $A \rightarrow_{\beta} B$ then $A \rightarrow_{\lambda \times} B$.

**Proof:** Induction on the structure of $A$. We treat the case $A \equiv (\lambda x. A_1) A_2, B \equiv A_1[A_2/x]$. Then $A \rightarrow_{\beta} A_1(x:=A_2) \rightarrow_{\lambda \times} x(A_1(x:=A_2)) \overset{Lemma 3.6}{=} x(A_1)[x(A_2)/x] \overset{\lambda \times}{=} A_1[A_2/x]$.

\[ \square \]

A final property of $\lambda x$ that can be shown easily is the confluence of $\rightarrow_{\lambda \times}$.

**Theorem 3.9** $\rightarrow_{\lambda \times}$ is confluent.

**Proof:** If $A \rightarrow_{\lambda \times} B_1$ and $A \rightarrow_{\lambda \times} B_2$ then by projection $x(A) \rightarrow_{\beta} x(B_i)$ so by confluence of $\rightarrow_{\beta}$ there is a pure term $C$ such that $B_i \rightarrow_{\beta} C$, now by soundness $x(B_i) \rightarrow_{\lambda \times} C$. Then also $B_1 \rightarrow_{\lambda \times} C$. \[ \square \]
4 The recursive path order

In this section we briefly introduce the recursive path order. For a more detailed description and proofs, the reader is referred to [Dershowitz 79], [Zantema 94] and [Ferreira & Zantema 94].

Definition 4.1 Let $F$ be a set of function symbols, $X$ a set of variables such that $F \cap X = \emptyset$, let $T(F, X)$ be the set of (open) terms over $F$ and $X$. Let $\succ$ be a partial order on $F$. Let $\tau$ be a map assigning to every function symbol $f \in F$ one of the words mult or lex.

The recursive path order $\succ_{\mathsf{rpo}}$ on $T(F, X)$ induced by $\succ$ and $\tau$ is defined by

$$f(s_1, \ldots, s_m) \succ_{\mathsf{rpo}} g(t_1, \ldots, t_n) \iff \exists i \leq m, s_i \succ_{\mathsf{rpo}} g(t_1, \ldots, t_n)$$

$$\lor (f \succ g \land \forall j[f(s_1, \ldots, s_m) \succ_{\mathsf{rpo}} t_j])$$

$$\lor (f = g \land \forall j[f(s_1, \ldots, s_m) \succ_{\mathsf{rpo}} t_j] \land \langle s_1, \ldots, s_m \rangle \succ_{\mathsf{rpo}} T(f) \langle t_1, \ldots, t_n \rangle)$$

Here $\succ_{\mathsf{lex}}$ and $\succ_{\mathsf{mult}}$ are respectively the lexicographic and the multiset extensions of $\succ_{\mathsf{rpo}}$, i.e.,

- $\langle s_1, \ldots, s_m \rangle \succ_{\mathsf{lex}}_{\mathsf{rpo}} \langle t_1, \ldots, t_n \rangle$ iff for some $i \leq m, n$, $s_i = t_i, s_{i-1} = t_{i-1}, s_i \succ_{\mathsf{rpo}} t_i$ or $s_1 = t_1, \ldots, s_m = t_m, m > n$.

- $\langle s_1, \ldots, s_m \rangle \succ_{\mathsf{mult}}_{\mathsf{rpo}} \langle t_1, \ldots, t_n \rangle$ iff the multiset $\{s_1, \ldots, s_m\}$ can be transformed into the multiset $\{t_1, \ldots, t_n\}$ by performing the operation 'replace a member $s$ of the multiset by finitely many terms $t$ such that $s \succ_{\mathsf{rpo}} t$ one or more times.'

In [Ferreira & Zantema 94], $\tau$ is called status function. More complex extensions of $\succ_{\mathsf{rpo}}$ than multiset or lexicographic are even possible.

Theorem 4.2 (Dershowitz) Let $\succ$ be a partial order and $\tau$ a status function on a set of function symbols $F$, let $\succ_{\mathsf{rpo}}$ be the induced recursive path order. Then

$\succ_{\mathsf{rpo}}$ is well-founded $\iff \succ$ is well-founded

Proof: see [Dershowitz 79] or [Ferreira & Zantema 94].

5 PSN for $\lambda x$

In this section we use the recursive path order to show that $\lambda x$ has PSN. Since the recursive path order is about first order term rewrite systems, we need to translate terms of $\lambda x$ into a first order term rewrite system; to be able to prove PSN this translation must in some sense preserve reductions. We do this by labelling (some) function symbols with maximal lengths of reduction sequences; therefore we restrict to terms where these lengths are finite for all subterms. It will turn out that these are exactly all the strongly normalizing $\lambda x$-terms.

Definition 5.1 We define the set $\lambda x_{<\infty} \subseteq \lambda x$ by

$$\lambda x_{<\infty} = \{A \in \lambda x \mid \text{for all subterms } B \text{ of } A, x(B) \in SN_{\beta}\}$$

Remark: $A \in \lambda x_{<\infty}$ if and only if: for all subterms $B$ of $A$, $\hat{\beta}(B) < \infty$.

Lemma 5.2 $\hat{\beta}(\lambda x. A)B > \hat{\beta}(A(x:=B))$.

Proof: Every $\rightarrow_{\beta}$-reduction path of length $n$ of $x(A(x:=B))$ can be extended to a reduction path of length $n + 1$ of $x((\lambda x. A)B)$ by prefixing it by $x((\lambda x. A))x(B) \rightarrow_{\beta} x(A)x(B)[x(B)/x] \equiv x(A(x:=B))$. \qed
Lemma 5.3 If $A \in \lambda x^<\infty$ and $A \rightarrow_{\lambda x} A'$ then $A' \in \lambda x^<\infty$.

Proof: Induction on the structure of $A$.

Definition 5.4 We define the set of labelled terms $\lambda^l$ by the following abstract syntax:

$$A ::= * | A \cdot n | \lambda A | A(A)_n$$

where $n$ ranges over the natural numbers.

We define by induction on the structure of terms a translation $T : \lambda x^<\infty \rightarrow \lambda^l$:

$$T(x) = *$$

$$T(AB) = T(A) \cdot n T(B) \quad \text{where } n = \beta(AB)$$

$$T(\lambda x.A) = \lambda T(A)$$

$$T(A(x:=B)) = T(A)(T(B)_n) \quad \text{where } n = \beta(A(x:=B))$$

Note that for all $A \in \lambda x^<\infty$, $T(A)$ is well-defined.

Definition 5.5 We define a TRS $\lambda^l$ using as terms the set $\lambda^l$ and having the reduction relation $\rightarrow_\tau$ defined by

$$(\lambda A)_m B \rightarrow_\tau A(B)_n \quad \text{if } m > n$$

$$(A \cdot m B)(C)_n \rightarrow_\tau (A(C)_p \cdot q (B(C)_r) \quad \text{if } n \geq p, q, r$$

$$(\lambda A)(C)_n \rightarrow_\tau \lambda(A(C)_n)$$

$$A(C) \rightarrow_\tau C$$

$$A(C) \rightarrow_\tau A \cdot n B \quad \text{if } m > n$$

$$A(B)_m \rightarrow_\tau A(B)_n \quad \text{if } m > n$$

Note that $\rightarrow_\tau$ is not confluent; for our purposes this is no problem since $\rightarrow_\tau$ is only designed to be useful for proving strong normalisation. The last two rules are called Decr in [Zantema 94] and are necessary to decrease the labels of applications and substitutions if inside of them a $\rightarrow_{\beta\eta}$-reduction is performed. Note that in the presence of the Decr rules we could also have $(\lambda A)_m^\tau B \rightarrow_\tau A(B)_n$ for all $n$ instead of $(\lambda A)_m B \rightarrow_\tau A(B)_n$ for all $m > n$.

Lemma 5.6 The relation $\rightarrow_\tau$ is a subrelation of some recursive path order. (That is, there is a precedence relation $\triangleright$ such that for all $A, B \in \lambda^l$, if $A \rightarrow_\tau B$, then $A \triangleright_{\text{rpo}} B$, where $\triangleright_{\text{rpo}}$ is the rpo ordering induced by $\triangleright$.)

Proof: Define the precedence $\triangleright$ by

$$\sim_m \triangleright \sim_n \triangleright \lambda \cdot *$$

and the status function $\tau$ by $\tau(\sim_m) = \tau(\lambda) = \tau(\sim_n) = \text{lex}$. Then $\rightarrow_\tau$ is a subrelation of the induced recursive path order $\triangleright_{\text{rpo}}$.

Corollary 5.7 $\rightarrow_\tau$ is SN.

Proof: By Theorem 4.2, $\triangleright_{\text{rpo}}$ of Lemma 5.6 is strongly normalising, hence by Lemma 5.6 $\rightarrow_\tau$ is strongly normalising.

Lemma 5.8 If $A \in \lambda x^<\infty$ and $A \rightarrow_{\lambda x} A'$ then $T(A) \rightarrow_\tau T(A')$.

Proof: Induction on the structure of $A$; we treat some of the more interesting cases.
• \( A \equiv (\lambda x.A_1)A_2 \rightarrow_{\text{Beta}} A_1(x:=A_2) \equiv A' \).

Then \( T(A) \equiv (\lambda T(A_1)) \rightarrow_m T(A_2) \rightarrow_\lambda T(A_1)(x:=T(A_2)) \equiv T(A') \) where \( m = \bar{\beta}(A) \); \( n = \bar{\beta}(A') \); note that \( m > n \) by Lemma 5.2.

• \( A \equiv (A_1 A_2)(x:=A_3) \rightarrow_\times (A_1(x:=A_3))(A_2(x:=A_3)) \equiv A' \).

Then \( T(A) \equiv (T(A_1) \rightarrow_m T(A_2)) \rightarrow_\lambda T(A_1)(T(A_3)) \rightarrow_n T(A_2)(T(A_3)) \equiv T(A') \), where \( m = \bar{\beta}(A_1 A_2) \), \( n = \bar{\beta}(A) \), \( p = \bar{\beta}(A_1(x:=A_3)) \), \( q = \bar{\beta}(A_2(x:=A_3)) \); note that \( n \geq p \) and \( n \geq q \).

• \( A \equiv x(x:=A_1) \rightarrow_\times A_1 \equiv A' \). Then \( T(A) \equiv *(T(A_1)) \rightarrow_\lambda T(A_1) \equiv T(A') \) where \( m = \bar{\beta}(A) \).

• \( A \equiv A_1(x:=A_2) \rightarrow_{\text{gc}} A_1 \equiv A' \). Then \( T(A) \equiv T(A_1)(T(A_2)) \rightarrow_\lambda T(A_1) \equiv T(A') \) where \( m = \bar{\beta}(A) \).

• \( A \equiv \lambda y.(A_1)(x:=A_2) \equiv A' \). Then \( T(A) \equiv (\lambda T(A_1))(T(A_2)) \rightarrow_\lambda (\lambda T(A_1))(T(A_2)_m) \rightarrow_\lambda T(A') \) where \( m = \bar{\beta}(A) = \bar{\beta}(A') \).

• \( A \equiv A_1 A_2 \rightarrow_{\text{lex}} A_1 A_2 \equiv A' \). Then \( T(A) \equiv T(A_1)(T(A_2)) \rightarrow_\lambda T(A_1)(T(A_2)) \rightarrow_\lambda T(A') \) where \( m = \bar{\beta}(A) \geq n = \bar{\beta}(A') \).

\( \square \)

Corollary 5.9 \((\text{PSN})\) 1. \( A \in SN_{\lambda x} \iff A \in \lambda x^{<\infty} \)

2. \( \lambda x \) preserves strong normalisation

6 \( \lambda \nu \), \( \lambda s \) and extensions

In this section we show that our method is general enough to show PSN for other calculi of explicit substitutions such as \( \lambda \nu \) of [BBLR 95] and \( \lambda s \) of [Kamareddine & Rios 95], and also some extensions of \( \lambda x \). Furthermore, we discuss some extensions of \( \lambda x \), giving a counterexample to PSN similar to the one of [Mellies 95], but less involved.

6.1 The calculi \( \lambda \nu \) and \( \lambda s \)

**Definition 6.1** Terms and substitutions of \( \lambda \nu \) are defined by the following abstract syntaxes:

\[
\begin{align*}
\text{a} & ::= n \mid (aa) \mid (\lambda a) \mid (a[s]) \\
\text{s} & ::= a[] \mid \uparrow(s) \mid \uparrow
\end{align*}
\]

where \( n \) ranges over the set \( \{1, 2, 3, 4, \ldots\} \).

The reduction relation \( \rightarrow_{\lambda \nu} \) is the union of \( \rightarrow_{\text{uBeta}} \) and \( \rightarrow_\nu \) which are defined by

\[
\begin{align*}
(\lambda a)b & \rightarrow_{\text{uBeta}} a[b/] \\
(ab)[s] & \rightarrow_\nu a[s][s] \\
(\lambda a)[s] & \rightarrow_\nu \lambda(a[\uparrow(s)]) \\
1[a/] & \rightarrow_\nu a \\
n + 1[a/] & \rightarrow_\nu n \\
1[\uparrow(s)] & \rightarrow_\nu 1 \\
n + 1[\uparrow(s)] & \rightarrow_\nu n[s][1] \\
n[1] & \rightarrow_\nu n + 1
\end{align*}
\]

7
Some initial intuition to motivate the reduction rules of $\lambda v$: $a[b/]$ stands for 'substitute $b$ for $1$ in $a'$, $[\uparrow (s)]$ stands for the substitution obtained by first raising all the indices in $s$ by $1$ and substituting not the index $1$, but the index $2$, and $[[ ]]$ stands for the substitution that raises all numbers (in the term in front of it) by $1$. An example to explain these intuitive motivations is the following. (For reasons of legibility we have removed some brackets.)

$$(\lambda(\lambda(12)))(11) \rightarrow_{\text{Beta}} (\lambda(12))[11/]$$

$$\rightarrow_{v} \lambda((12)[\uparrow (11/)])$$

$$\rightarrow_{v} \lambda(1[\uparrow (11/)]2[\uparrow (11/)])$$

$$\rightarrow_{v} \lambda(12[\uparrow (11/)])$$

$$\rightarrow_{v} \lambda(1(11[\uparrow (1)])$$

$$\rightarrow_{v} \lambda(1(1[1]1[1]))$$

$$\rightarrow_{v} \lambda(1(122))$$

For a detailed explanation and motivation of the system $\lambda v$ we refer to [BBLR 95].

**Definition 6.2** Terms and substitutions of $\lambda s$ are defined by the following abstract syntaxes:

$$a ::= n \mid (aa) \mid (\lambda a) \mid (\phi_{j} a) \mid (as^{i}a)$$

where $n, i$ range over the set $\{1, 2, 3, 4, \ldots \}$ and $j$ ranges over $\{0, 1, 2, 3, \ldots \}$.

The reduction relation $\rightarrow_{\lambda s}$ is the union of $\rightarrow_{\text{Beta}}$ and $\rightarrow_{s}$ which are defined by

$$(\lambda b) \rightarrow_{\text{Beta}} \lambda(s^{i}b)$$

$$(\lambda a)s^{i}b \rightarrow_{s} \lambda(as^{i+1}b)$$

$$(a_{1}a_{2})s^{i}b \rightarrow_{s} (a_{1}s^{i}b)(a_{2}s^{i}b)$$

$$n s^{i}b \rightarrow_{s} \begin{cases} n - 1 & \text{if } n > i \\ \phi_{0}(b) & \text{if } n = i \\ n & \text{if } n < i \end{cases}$$

$$\phi_{k}^{i}(\lambda a) \rightarrow_{s} \lambda(\phi_{k}^{i+1}a)$$

$$\phi_{k}^{i}(a_{1}a_{2}) \rightarrow_{s} (\phi_{k}^{i}a_{1})(\phi_{k}^{i}a_{2})$$

$$\phi_{k}^{i}n \rightarrow_{s} \begin{cases} n + i - 1 & \text{if } n > k \\ n & \text{if } n \leq k \end{cases}$$

Again, we don't give a detailed explanation and motivation for the rules of this calculus, but refer to [Kamareddine & Rios 95]. Some initial intuition: $s^{i}(b)$ stands for the substitution of $b$ for $i$, $\phi_{k}^{i}(a)$ stands for 'raise all the numbers $n > k$ in the term $a$ with $i - 1$'. To explain the rules, we treat the same example as for $\lambda v$.

$$(\lambda(\lambda(12)))(11) \rightarrow_{\text{Beta}} (\lambda(12))s^{1}(11)$$

$$\rightarrow_{s} \lambda((12)s^{2}(11))$$

$$\rightarrow_{s} \lambda(1s^{2}(11))(2s^{2}(11)))$$

$$\rightarrow_{s} \lambda(1(2s^{2}(11)))$$

$$\rightarrow_{s} \lambda(1(\phi_{0}^{1}(1)))$$

$$\rightarrow_{s} \lambda(1(\phi_{0}^{1}(1))\phi_{0}(1))$$

$$\rightarrow_{s} \lambda(1(22))$$
The calculus $\lambda v$ is very similar to $\lambda s$. The difference is mainly in the moment of updating: in $\lambda v$ every step $n + 1[f(t(s))] \rightarrow_v z[s][1]$ creates an update substitution $[1]$ whereas in $\lambda s$ the update function symbol $\phi_0^s$ is only created at the actual moment of substitution in $n \sigma^a \rightarrow \phi^s_0 a$. Also, in the reductions $u \sigma^1 b \rightarrow_n n - 1$ ($n > i$) and $u \sigma^0 b \rightarrow_n n$ ($n < i$), there is no update function generated whereas in $n + 1[f(t(s))] \rightarrow_v n[s][1]$ an update substitution is created regardless of whether the substitution $[1]$ is binding or is void.

In [BBLR 95] it is shown that $\lambda v$ has PSN by contradicting the existence of a minimal infinite $\lambda v$-reduction of a term which is SN for $\rightarrow_{\beta}$; in [Kamareddine & Rios 95] PSN is shown to hold for $\lambda s$ in a similar way.

We show that $\lambda v$ and $\lambda s$ are PSN by using the labelled calculus $\lambda I$. The proof is very similar to the proof of PSN for $\lambda x$ that we gave in the previous section.

For $\lambda v$ and $\lambda s$ we have the usual properties such as SN, CR, UN for $\rightarrow_v$ respectively $\rightarrow_s$, substitution lemma, projection lemma, soundness lemma and confluence for $\rightarrow_{\lambda v}$ respectively $\rightarrow_{\lambda s}$. We denote the $\rightarrow_v$-normal form respectively $\rightarrow_s$-normal form of a term $b$ by $v(b)$ respectively $s(b)$.

Note that a substitution of $\lambda v$ is of the form $n \sigma^a \rightarrow \phi_0^a a$ for some $n$.

We denote $\beta$-reduction on $\lambda v$-terms as well as on $\lambda s$-terms by $\rightarrow_{\beta}$; for a $\lambda v$-respectively $\lambda s$-term $a$ we write $\beta(a)$ to denote the maximal number of $\beta$-reduction steps starting from $v(a)$ respectively $s(a)$, if this number exists.

Definition 6.3

$$\lambda v<\infty := \{ a \in \lambda v \mid \forall b \subseteq a[v(b) \in SN_{\beta}\}\}$$

$$\lambda s<\infty := \{ a \in \lambda s \mid \forall b \subseteq a[s(b) \in SN_{\beta}\}\}$$

Lemma 6.4

1. $\lambda v<\infty$ is closed under $\rightarrow_{\lambda v}$-reduction

2. $\lambda s<\infty$ is closed under $\rightarrow_{\lambda s}$-reduction

Definition 6.5

1. $T_v : \lambda v<\infty \rightarrow \Lambda^I$ is defined by

$$T_v(n) = *$$

$$T_v(ab) = T_v(a)_p T_v(b) \quad \text{where} \quad p = \beta(ab)$$

$$T_v(\lambda a) = \lambda T_v(a)$$

$$T_v(a[\eta^n (b/)])) = T_v(a)(T_v(b))_p \quad \text{where} \quad p = \beta(a[\eta^n (b/)]))$$

$$T_v(a[\eta_p (t)]) = T_v(a)$$

2. $T_s : \lambda s<\infty \rightarrow \Lambda^I$ is defined by

$$T_s(n) = *$$

$$T_s(ab) = T_s(a)_p T_s(b) \quad \text{where} \quad p = \beta(ab)$$

$$T_s(\lambda a) = \lambda T_s(a)$$

$$T_s(a[\phi_0^a (b/)])) = T_s(a)(T_s(b))_p \quad \text{where} \quad p = \beta(a[\phi_0^a (b/)]))$$

$$T_s(\phi_0^a a) = T_s(a)$$

Lemma 6.6

1. If $a \in \lambda v<\infty$ and $a \rightarrow_v b$ then $T_v(a) \rightarrow_i T_v(b)$

2. If $a \in \lambda v<\infty$ and $a \rightarrow_{\beta} b$ then $T_v(a) \rightarrow_i T_v(b)$

3. If $a \in \lambda s<\infty$ and $a \rightarrow_s b$ then $T_s(a) \rightarrow_i T_s(b)$

4. If $a \in \lambda s<\infty$ and $a \rightarrow_{s Beta} b$ then $T_s(a) \rightarrow_i T_s(b)$


Theorem 6.7

1. $a \in SN_{\lambda v} \iff a \in \lambda v<\infty$
2. $\rightarrow_{\lambda u}$ has PSN
3. $a \in SN_{\lambda u} \iff a \in \lambda b^0<\infty$
4. $\rightarrow_{\lambda s}$ has PSN

Proof:
1. $\Rightarrow$ by projection: $\Leftarrow$: since $\rightarrow_v$ is SN, any infinite $\rightarrow_{\lambda v}$-reduction must contain infinitely many $\rightarrow_{\beta \text{Beta}}$-steps. Therefore an infinite reduction of a pure term which is SN for $\rightarrow_{\beta}$ translates by $T_v$ into an infinite $\rightarrow_{\beta}$-reduction which is impossible by 5.7.
2. follows from 1.
3. & 4. similar to 1. & 2.

6.2 Extensions of $\lambda x$

In this section we consider several extensions of $\lambda x$ with some kind of composition. The calculus $\lambda \sigma$ of [Abadi et al. 90] was designed to be able to compose substitutions. The price however is not having PSN (cf. [Mellies 95]). Since $\lambda x$ has no composition but does have PSN, it is an interesting question where the borderline is between PSN and composition of substitutions.

We start with a short discussion of $\lambda \sigma$. For the precise definition of $\lambda \sigma$, the reader is referred to [Abadi et al. 90]. The composition of substitutions in $\lambda \sigma$ is mainly performed by two rules, Comp and Map. The first glues two substitutions together: $a[s][t] \xrightarrow{\text{Comp}} a[sot]$, while Map allows the distribution of the second substitution over the first: $(b \cdot c \cdot s')ot \xrightarrow{\text{Map}} b[t] \cdot ((c \cdot s')ot) \xrightarrow{\text{Map}} b[t] \cdot c[t] \cdot (s'ot)$.

As was pointed out in [Kamareddine & Nederpelt 93], the substitutions of $\lambda \sigma$ are roughly the same as simultaneous parallel substitutions in the following extension of $\lambda x$:

$$\text{terms } t ::= x \mid tt \mid \lambda x.t \mid t(i:=t)$$

where \(\langle x:=t \rangle\) is shorthand for \((x_1, \ldots, x_m:=t_1, \ldots, t_m)\); reductions are similar as for $\lambda x$ plus the composition rule

$$a(x:=b)(y:=c) \rightarrow a(x:=b(y:=c)) \quad \text{if } y \notin \text{FV}(a)$$

(no simultaneous substitutions needed) doesn't have PSN. We give an infinite derivation starting from the term $(\lambda y.(\lambda y'.x)((\lambda y.a)b))((\lambda y.a)b)$. Note that this term is even simpler than the term used in [Mellies 95].

First we define substitutions $\alpha_m$ for $m \in \mathbb{N}$ by

$$\alpha_0 = (y:=((\lambda y.a)b))$$
$$\alpha_{m+1} = (y:=b\alpha_m)$$

Now consider the following three reductions. (For simplicity we forget about the variable convention during this counterexample; furthermore, we freely change bound variables if convenient.)

$$(\lambda y.(\lambda y'.x)((\lambda y.a)b))((\lambda y.a)b) \rightarrow (\lambda y'.(\lambda y.a)b)(\lambda y.a)b)$$
$$\rightarrow (\lambda y'.((\lambda y.a)b)(\lambda y.a)b))$$
$$\rightarrow (\lambda y'.(\lambda y.a)(\lambda y.a)b)(b(\lambda y.a)b))$$
$$\equiv (\lambda y'.(\lambda y.a\alpha_0)(\lambda y.a)b))$$
$$\rightarrow (\lambda y'.a\alpha_0(\lambda y.a\alpha_1))$$
$$\equiv \lambda y'.(\lambda y.a\alpha_0)(\lambda y.a\alpha_1)$$

\[\square\]
\[ a\alpha_0\alpha_{m+1} \equiv a(y':=(\lambda y.a)b)(y:=b\alpha_m) \rightarrow a(y':=(\lambda y.a)b)(y:=b\alpha_m) \]  
\[ \rightarrow a(y':=(\lambda y.a)(y:=b\alpha_m))(b(y:=b\alpha_m)) \equiv a(y':=(\lambda y.a\alpha_{m+1})(b\alpha_{m+1})) \]  
\[ \rightarrow a(y':=a\alpha_{m+1}(y:=b\alpha_{m+1})) \equiv a(y':=a\alpha_{m+1}\alpha_{m+2}) \]

\[ a\alpha_{m+1}\alpha_{n+1} \equiv a(y':=b\alpha_m)(y:=b\alpha_n) \rightarrow a(y':=b\alpha_m(y:=b\alpha_n)) \equiv a(y':=b\alpha_m\alpha_{n+1}) \]

These combine into an infinite derivation in the following way.

\[ (\lambda y.(\lambda y'.x)((\lambda y.a)b))(\lambda y.a)b) \rightarrow \]  
\[ ... \rightarrow \ldots a\alpha_0\alpha_1 \ldots \rightarrow \ldots a\alpha_0\alpha_2 \ldots \rightarrow \ldots a\alpha_0\alpha_3 \ldots \]  
\[ \vdots \]  
\[ \rightarrow \ldots a\alpha_0\alpha_{m+1} \ldots \rightarrow \ldots a\alpha_{m+1}\alpha_{m+2} \ldots \rightarrow \ldots a\alpha_0a\alpha_2 \ldots \]

Recall that we proved PSN by showing that every term of which the x-normal form of any of its subterms is in SN_\beta, is also SN for \beta \rightarrow \_x. With the extra composition reduction defined above, there is an easy counterexample to that: the term \( x(y:=zZ)(Z:=AW.ww) \) has x-normalform \( x \) (and is also SN for \( \rightarrow_{Ax} \)) but it has Omega as a subterm of a reduct if composition is allowed.

This example also shows why our method fails for the system extended with the extra composition rule, and hence also for \( \lambda x: T(x(y:=zZ)(Z:=AW.ww)) \equiv \ast(\ast \ast \ast)(\lambda(\ast \ast \ast))_0 \) whereas after composition of the two substitutions, the label of the innermost substitution does not exist: \( T(x(y:=(xZ))(Z:=\lambda w.ww)) \equiv \ast(\ast \ast \ast)(\lambda(\ast \ast \ast))_0 \). So reduction in \( \lambda x \) does not always decrease T-images.

One can try to give a rule for composition of substitutions such that reduction still decreases T-images, the following rule seems best fit for this purpose:

\[ a(x:=b)(y:=c) \rightarrow_x a(x:=b(y:=c)) \text{ if } y \notin FV(a), x \in FV(x(a)) \]

The idea behind this rule is that, if \( x \in FV(x(a)) \), then \( b(y:=c) \) will occur as a subterm of some \( \rightarrow_{Ax} \)-reduct. Hence allowing to create \( b(y:=c) \) at this point will not spoil PSN. We strongly believe that adding this reduction rule does not spoil SN (i.e. \( \lambda x \) extended with the rule \( \rightarrow_{Ax} \) has PSN), but we have not been able to prove it.

7 Proof of PSN using labelled trees

In this section we outline a proof of the Preservation of Strong Normalization property, again using the RPO technique, but now in the way it has been presented in [Klop 92]. One then looks at the collection of commutative finite labelled trees \( \text{Tree} \) (i.e. trees are identified up to permutation of branches; there is no order from left to right in the subtrees). The labels are taken from \( N \). Furthermore, one looks at the set \( \text{Tree}^* \), where some nodes in a tree may have a marker \( \ast \). It is convenient to denote the tree with root node \( n \) and subtrees \( t_1, \ldots, t_p \) by \( n(t_1, \ldots, t_p) \), and similarly, if the root node has a marker, by \( n^*(t_1, \ldots, t_p) \). In the following, we abbreviate \( t_1, \ldots, t_p \) to \( \vec{t} \). On these commutative labelled trees with markers (the set \( \text{Tree}^* \)), a reduction relation \( \Rightarrow \) is defined.

**Definition 7.1** The relation \( \Rightarrow \) on \( \text{Tree}^* \) is defined as follows.

\[ n(\vec{t}) \Rightarrow n^*(\vec{t}), \]
\[ n^*(\vec{t}) \Rightarrow m(n^*(\vec{t}), \ldots, n^*(\vec{t})), \]
\[ n^*(s, \vec{t}) \Rightarrow n(s^*, \ldots, s^*, \vec{t}), \]
\[ n^*(\vec{t}) \Rightarrow t_i, \]
\[ 1 \leq i \leq p. \]

11
Furthermore, the relation $\Rightarrow$ is compatible with the tree-forming operations, that is, if $t_i \Rightarrow t'_i$, then $n(t_1, \ldots, t_i, \ldots, t_p) \Rightarrow n(t'_1, \ldots, t'_i, \ldots, t'_p)$.

As usual, the relation $\Rightarrow^+$ denotes the transitive closure of $\Rightarrow$ and $\Rightarrow^*$ denotes the transitive reflexive closure of $\Rightarrow$.

For examples on the use of these rules we refer to [Klop 92], we just mention the main result, which will be applied here to the issue of PSN for explicit substitution.

**Theorem 7.2 ([Klop 92],[Dershowitz 79])** The relation $\Rightarrow^+$ is well-founded on $\text{Tree}$ (the set of trees without markers).

To prove PSN for the calculus $\lambda x$, we now proceed by defining a reduction preserving mapping $T$ from $\lambda x^{<\infty}$ to $\text{Tree}$: if $M \rightarrow_{\lambda x} N$, then $M \Rightarrow^* N$ and if $M \rightarrow_{\lambda x} N$, then $M \Rightarrow^+ N$. Hence, using the fact that $\rightarrow_{\lambda x}$ is strongly normalizing, we can again conclude that every $M \in \lambda x^{<\infty}$ is strongly normalizing and so that $\lambda x$ has the PSN property.

For notational convenience, we abbreviate the sequence of definitions $(x_1 := P_1) \cdots (x_n := P_n)$ to $(x := P)$.

**Definition 7.3** For $M \in \lambda x^{<\infty}$, we define the tree $T(M)$ by induction on the length of $M$ as follows.

$$
T(x) = 0,
$$

$$
T(QN) = \begin{cases}
T(Q) & \text{if } y \notin \{x_1, \ldots, x_n\} \\
T(N) & \text{otherwise}
\end{cases}
$$

$$
T(\lambda y. N) = T(N)
$$

$$
T(y(x := P)) = \begin{cases}
T(P_1) & \text{if } y \notin \{x_1, \ldots, x_n\} \\
\ldots & \\
T(P_n) & \text{otherwise}
\end{cases}
$$

$$
T(x_i(x := P)) = \begin{cases}
T(P_1) & \text{if } y \notin \{x_1, \ldots, x_n\} \\
\ldots & \\
T(P_{i-1}) & \text{if } y \notin \{x_1, \ldots, x_n\} \\
T(P_i(x_{i+1} := P_{i+1}) \cdots (x_n := P_n)) & \text{otherwise}
\end{cases}
$$

$$
T((QN)(x := P)) = \begin{cases}
T(Q(x := P)) & \text{if } y \notin \{x_1, \ldots, x_n\} \\
T(N(x := P)) & \text{otherwise}
\end{cases}
$$

$$
T((\lambda y. N)(x := P)) = T(N(x := P))
$$
The following Lemmas show that $T$ preserves reductions (in the right sense as announced above). The proofs of these Lemmas are not difficult, the main complication being to find out the right induction loadings (and the right order on which the induction should be done). We just outline the proofs.

**Lemma 7.4** For $M \in \lambda x^{<\infty}$, if $M \rightarrow^* N$, then $T(M) \rightarrow^* T(N)$.

**Proof:** By induction on the length of $M$, distinguishing subcases according to the structure of $M$. Note that we need Lemma 5.3 to make sure that $N \in \lambda x^{<\infty}$ and hence that $T(N)$ is well-defined. □

The following two Lemmas are sublemmas necessary for the proof of preservation of $\rightarrow_{\text{Beta}}$-reduction by $T$.

**Lemma 7.5** For $N(x:=P) \in \lambda x^{<\infty}$, $T(N(x:=P)) \rightarrow^* T(N)$.

**Proof:** By induction on the length of $N$. □

**Lemma 7.6** For $(\lambda y.N)(x:=P) \in \lambda x^{<\infty}$, $T((\lambda y.N)(x:=P)) \rightarrow^+ T(N(y:=Q)(x:=P))$.

**Proof:** By induction on the length of $N$, using Lemma 7.5. First write $N$ as $R(y:=Q)$, with $R$ not a term that ends with a substitution item. (So, the sequence $(y:=Q)$ should be taken as long as possible.) Then distinguish cases according to the structure of $R$. □

**Corollary 7.7** The calculus $\lambda x$ has the PSN property.

## 8 Conclusions

We have introduced a new method for proving PSN of lambda calculi with explicit substitution. The method involves four steps:

- determine a suitable set contained in the set of strongly normalising terms in the explicit substitution calculus, containing the pure $\beta$-SN terms and closed under explicit substitution reduction,
- give a translation from this set into a first order term rewrite system,
- define a strongly normalising reduction relation on this TRS by giving a well-founded precedence,
- show that the translation preserves infinite reduction paths.

For named calculi, the translation identifies all variables; for calculi using de Bruijn indices the translation identifies all indices and erases update functions, giving evidence for the statement ‘update functions do not matter for termination issues’. Kruskal’s theorem ensures that a well-founded precedence yields a strongly normalizing term rewrite system.

Further applications of this method that are under investigation:

- give a maximal strategy for $\lambda x$-reduction and an inductive characterization of the set $\lambda x^{<\infty}$,
- give a general PSN proof for combinator reduction systems with explicit substitution (cf. [Rose 95], [Bloo & Rose 96])
- give a (first order) calculus with explicit substitution which has PSN as well as confluence on open terms.
9 Acknowledgements

Thanks to Thomas Arts for making us aware of current notations for the recursive path orders, using semantic labelling. We have also benefited from discussions with the following people: Hans Zantema, Gilles Barthe, Daniel Briaud, Twan Laan, Pierre Lescanne, Rob Nederpelt and Kristoffer Rose.

References


<table>
<thead>
<tr>
<th>Computing Science Reports</th>
<th>Department of Mathematics and Computing Science</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In this series appeared:</strong></td>
<td><strong>Eindhoven University of Technology</strong></td>
</tr>
<tr>
<td>93/01 R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.</td>
</tr>
<tr>
<td>93/02 T. Verhoeff</td>
<td>A continuous version of the Prisoner's Dilemma, p. 17</td>
</tr>
<tr>
<td>93/03 T. Verhoeff</td>
<td>Quicksort for linked lists, p. 8</td>
</tr>
<tr>
<td>93/04 E.H.L. Aarts, J.H.M. Korst, P.J. Zweliering</td>
<td>Deterministic and randomized local search, p. 78.</td>
</tr>
<tr>
<td>93/05 J.C.M. Baeten, C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates, p. 18.</td>
</tr>
<tr>
<td>93/06 J.P. Veltkamp</td>
<td>On the unavoidability of metastable behaviour, p. 29</td>
</tr>
<tr>
<td>93/07 P.O. Moerland</td>
<td>Exercises in Multiprogramming, p. 97</td>
</tr>
<tr>
<td>93/08 J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.</td>
</tr>
<tr>
<td>93/10 K.M. van Hee</td>
<td>Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.</td>
</tr>
<tr>
<td>93/16 H. Schepers, J. Hoornaar</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27.</td>
</tr>
<tr>
<td>93/17 D. Aistein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>93/18 C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19 G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21.</td>
</tr>
<tr>
<td>93/22 E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23 E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24 E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26 W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27 T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28 F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29 R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30 J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
</tbody>
</table>
W. Körver

Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.

H. ten Eikelder and H. van Geldrop


L. Loyens and J. Moonen

ILIAS, a sequential language for parallel matrix computations, p. 20.

J.C.M. Baeten and J.A. Bergstra

Real Time Process Algebra with Infinitesimals, p. 39.

W. Ferrer and P. Severi

Abstract Reduction and Topology, p. 28.

J.C.M. Baeten and J.A. Bergstra

Non Interleaving Process Algebra, p. 17.

J. Brunakreef J-P. Katoen R. Koymans S. Mauw

Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

C. Verheof

A general conservative extension theorem in process algebra, p. 17.

W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Teulman Kip K.M. van Hee

Job Shop Scheduling by Constraint Satisfaction, p. 22.

P.D.V. van der Stok M.M.M.P.J. Claessen D. Aksteijn


A. Bijlsma

Temporal operators viewed as predicate transformers, p. 11.

P.M.P. Rambags

Automatic Verification of Regular Protocols in P/T Nets, p. 23.

B.W. Watson

A taxonomy of finite automata construction algorithms, p. 87.

B.W. Watson

A taxonomy of finite automata minimization algorithms, p. 23.

E.J. Luit J.M.M. Martin

A precise clock synchronization protocol, p.

T. Klou D. Kratsch J. Spinrad


W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky


R. Gerth

Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart

The object-oriented paradigm, p. 28.

F. Kamareddine R.P. Nederpelt

Canonical typing and β-conversion, p. 51.

L.B. Hamman K.M. van Hoe


J.C.M. Baeten J.A. Bergstra

Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.

P. Zhou J. Hoeman


T. Basten T. Kanz J. Black M. Coffin D. Taylor

Time and the Order of Abstract Events in Distributed Computations, p. 29.

K.R. Apt R. Bol


O.S. van Roosmalen

A Hierarchical Diagrammatic Representation of Class Structure, p. 22.

J.C.M. Baeten J.A. Bergstra

Process Algebra with Partial Choice, p. 16.
<table>
<thead>
<tr>
<th>Page</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>94/10</td>
<td>T. Verhoeven</td>
<td>The testing Paradigm Applied to Network Structure. p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljé</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peermans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doornbos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
<tr>
<td>94/18</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Refining Reduction in the Lambda Calculus, p. 15.</td>
</tr>
<tr>
<td>94/19</td>
<td>B.W. Watson</td>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
</tr>
<tr>
<td>94/20</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Beyond β-Reduction in Church's λ→, p. 22.</td>
</tr>
<tr>
<td>94/22</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.</td>
</tr>
<tr>
<td>94/23</td>
<td>S. Mauw and M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>94/25</td>
<td>T. Kloks</td>
<td>K,γ-free and Wγ-free graphs, p. 10.</td>
</tr>
<tr>
<td>94/29</td>
<td>J. Hoorman</td>
<td>Correctness of Real Time Systems by Construction, p. 22.</td>
</tr>
<tr>
<td>94/30</td>
<td>J.C.M. Baeten, J.A. Bergstra, Gh. Ştefănescu</td>
<td>Process Algebra with Feedback. p. 22.</td>
</tr>
<tr>
<td>94/31</td>
<td>B.W. Watson, R.E. Watson</td>
<td>A Boyer-Moore type algorithm for regular expression pattern matching, p. 52.</td>
</tr>
<tr>
<td>94/33</td>
<td>T. Laan</td>
<td>A formalization of the Ramified Type Theory, p.40.</td>
</tr>
<tr>
<td>94/35</td>
<td>J.C.M. Baeten, S. Mauw</td>
<td>Delayed choice: an operator for joining Message Sequence Charts, p. 15.</td>
</tr>
<tr>
<td>94/36</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Canonical typing and II-conversion in the Barendregt Cube, p. 19.</td>
</tr>
<tr>
<td>94/38</td>
<td>A. Bijlsma, C.S. Scholten</td>
<td>Point-free substitution, p. 10.</td>
</tr>
</tbody>
</table>
On the equivalence covering number of splitgraphs, p. 4.

Distributed Consensus and Hard Real-Time Systems, p. 34.

Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.

Concatenation of Graphs, p. 7.

Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.

Verifying Sequentially Consistent Memory, p. 160

The A-cube with classes of terms modulo conversion, p. 16.

On \( \Pi \)-conversion in Type Theory, p. 12.

Fixed-Point Calculus, p. 11.

Process Algebra with Propositional Signals, p. 25

A short and flexible proof of Strong Normalization for the Calculus of Constructions, p. 27.

Listing simplicial vertices and recognizing diamond-free graphs, p. 4.

Traces and Logic, p. 81

A Partial Order Approach to Branching Time Logic Model Checking, p. 20.

The Construction of a small Communication Library, p. 16.

Formalizing Process Algebraic Verifications in the Calculus of Constructions, p. 49.

Concrete process algebra, p. 134.

An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.

A Type Inference Algorithm for Pure Type Systems, p. 20.

A Quantitative Analysis of Iterated Local Search, p. 23.

Drawing Execution Graphs by Parsing, p. 10.

Preservation of Strong Normalisation for Explicit Substitution, p. 12.

Discrete Time Process Algebra, p. 20

MathJpad: A System for On-Line Preparation of Mathematical Documents, p. 15
Deductive Database Systems and integrity constraint checking, p. 36.
Empty interworkings and refinement
Semantics of interworkings Revised, p. 19.
De proceedings: ACP'95, p.
On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.
Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.
Specification of tools for Message Sequence Charts, p. 36.
On Normalisation, p. 33.
Axiomatizing Early and Late Input by Variable Elimination, p. 44.
Petri net based scheduling, p. 20.
Synchronous Sequence Charts In Action, p. 36.
A Class of Petri nets for modeling and analyzing business processes, p. 24.
A Conservative Look at term Deduction Systems with Variable Binding, p. 29.
Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions, p. 17.
Context-Free Graph Grammars and Concatenation of Graphs, p. 35.
Record concatenation with intersection types, p. 46.
An algebraic semantics for hierarchical P/T Nets, p. 32.
Process Algebra with Autonomous Actions, p. 12.
Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
Example specifications in phi-SDL.
A Process-Algebraic Approach to Life-Cycle Inheritance
Inheritance = Encapsulation + Abstraction, p. 15.
Life-Cycle Inheritance
A Petri-Net-Based Approach, p. 18.
Structural Petri Net Equivalence, p. 16.