

Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection

Roel Bloo *bloo@win.tue.nl*^{*} Kristoffer H. Rose *kris@diku.dk*[†]

Abstract

In this paper we introduce and study a new λ -calculus with explicit substitution, λxgc , which has two distinguishing features: first, it retains the use of traditional *variable names*, specifying terms modulo renaming; this simplifies the reduction system. Second, it includes reduction rules for *explicit garbage collection*; this simplifies several proofs.

We show that λxgc is a conservative extension which *preserves strong normalisation* (PSN) of the untyped $\lambda\beta$ -calculus. The result is obtained in a modular way by first proving it for garbage-free reduction and then extending to ‘reductions in garbage’. This provides insight into the counterexample to PSN for $\lambda\sigma$ of Mellès (1995); we exploit the abstract nature of λxgc to show how PSN is in conflict with any reasonable substitution composition rule (except for trivial composition rules of which we mention one).

Key words: lambda calculus, explicit substitution, strong normalisation, garbage collection.

1 Introduction

We commence by explaining how explicit substitution calculi originated as refinements of the original λ -calculus and why this refinement is considered useful. We then proceed with a brief presentation of the PSN (preservation of strong normalisation of λ -terms) property and why it is difficult to obtain. Finally we give an overview of the following sections.

The reader is expected to be familiar with the untyped λ -calculus (Barendregt 1984) and we make free use of abstract reduction system concepts (Huet 1980, Klop 1987), in particular the ‘diagram stencils’ of Rosen (1973). Due to space limitations, most proofs are omitted; the full paper includes detailed proofs and will be available as a technical report.

Infinity of the lambda calculus. Recall that the essential definition for reduction in λ -calculus is β -reduction $(\lambda x.M)N \rightarrow_{\beta} M[x := N]$ where $M[x := N]$ is metanotation for the result of replacing in M all (unbound) occurrences of x with N . What this really means is that the above ‘rule schema’ is really an *infinite family of explicit rewrite rules* indexed by the structure of λ -bodies M , *i.e.*, $(\lambda x.x)N \rightarrow_{\beta} N$, $(\lambda x.y)N \rightarrow_{\beta} y$ (if $x \neq y$), $(\lambda x.\lambda y.y)N \rightarrow_{\beta} \lambda y.y$, $(\lambda x.\lambda y.x)N \rightarrow_{\beta} \lambda y.N$ (if $x \neq y$), \dots

The need for computational models. In the sixties it was realised that the above infinity of rules, or rather: the fact that a given reduction could have consequences on arbitrarily large subterms, means that the λ -calculus is not a very nice computational model in the sense that the number of β -reduction steps required to compute something does not correspond well to the number of computation steps needed by any reasonable ‘real’ computer. This led to the study of *abstract machines*, notably the SECD machine (Landin 1964), and later to the study of *combinator reduction* (Turner 1979) that both satisfy the need for an elementary reduction that reflects a realistic measure of complexity. The price, however, is that the evaluation order is fixed, so it is difficult to reason about the equivalence of programs and correctness of program transformations because they typically involve reductions that are not allowed inside the system itself.

In the following decades this problem was recognised: modern explicit substitution calculi with elementary reduction without a predetermined reduction strategy, but instead designed to be confluent,

^{*}Eindhoven University of Technology, PO-Box 513, NL-5600 MB Eindhoven.

[†]DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 København Ø.

have been invented, notably $\lambda\sigma$ of Abadi, Cardelli, Curien and Lévy (1991) developed from Curien's (1986) Categorical Combinators.

In recent years several explicit substitution calculi have been reported (*cf.* Rose 1993, Kamareddine and Nederpelt 1993, Ríos 1993, Lescanne 1994). For several of these, abstract machines have been constructed based on the reduction steps of the calculus (*cf.* Curien 1990, Crégut 1990, Abadi et al. 1991, Benaissa, Briaud, Lescanne and Rouyer-Degli 1995).

Too many reductions! However, it turns out that there are too many possible ways to reduce with the traditional explicit substitution calculi in a certain sense: Melliès (1995) observed that $\lambda\sigma$ is in fact so liberal in its reduction principle that β -strongly normalising and even typable terms might have infinite reductions in $\lambda\sigma$: the main counterexample is the typable λ -term $\lambda v.(\lambda x.I(Ix))(Iv)$, where $I \equiv \lambda y.y$, whose representation in $\lambda\sigma$ has infinite reductions. This is unfortunate as one of the purposes of establishing confluence was to permit arbitrary reductions, *e.g.*, in order to perform program transformation, and this is not safe when there is a risk of unexpected infinite reduction.

Recovering preservation of strong normalisation (PSN). Since the announcement of Melliès's result several similar explicit substitution calculi have been shown to possess PSN (Benaissa et al. 1995, Bloo 1995, Lescanne and Rouyer-Degli 1995, Kamareddine and Ríos 1995). The calculus described in this paper was developed independently and is shown to have the PSN property. Our proof is constructive whereas the proofs in the aforecited papers are all highly nonconstructive.

Overview of this paper. In section 2 we present the λxgc -calculus with 'naïve' reductions for explicit substitution and garbage collection (it is naïve because we just make syntactic the standard definition of substitution) and show that this is a conservative extension of the ordinary λ -calculus (as should be expected) from which several properties, notably confluence, follow.

In section 3 we first consider the possibilities for extending λxgc by reintroducing composition of substitutions. The result, possibly disappointing, is that composition of substitutions is only allowed if no substitutions can be 'shifted' into garbage – a rather strong restriction. We then give the proof of PSN. Sacrificing a bit of the explicitness by making garbage collection implicit in the reduction enables us to give a simple proof of PSN even though substitution is still done explicitly and without restrictions on the reduction order. We then make garbage collection explicit again and show that PSN still holds by distinguishing different degrees of 'garbage reduction'.

Finally we conclude.

2 Explicit Substitution and Garbage Collection

This section is about the λxgc -calculus: an *explicit substitution* calculus in the tradition of $\lambda\sigma$ (Abadi et al. 1991). However, we have chosen to make the calculus simpler by including only 'naïve direct substitution' following Rose (1993), which makes our calculus closer to λv of Lescanne (1994) in that no composition of substitutions is added. Furthermore, the calculus has an explicit *garbage collection* rule which makes the proofs for some properties simpler (the used notion of garbage collection is from Rose (1993)).

We first present the terms we will use throughout the paper, and generalise the standard related concepts from the λ -calculus to it. Then we present a 'raw' notion of λxgc -reduction where all substitution and garbage collection steps are explicit. We then analyse properties of the substitution and garbage collection subrelations and explain the relation to standard substitution and β -reduction, showing that the reduction relation of λxgc is a conservative extension of $\overrightarrow{\beta}$ from which confluence follows.

Definition 2.1 (terms). The set of λxgc -terms Λx is ranged over by the letters $MNPQ$ and R , and defined inductively by $M ::= x \mid \lambda x.M \mid MM \mid M\langle x := M \rangle$ where the letters $xyzvw$ range over an infinite set of variables. As usual we use parentheses to indicate subterm structure and write $\lambda xy.M$ for $\lambda x.(\lambda y.M)$ and MNP for $(MN)P$; explicit substitution is given highest precedence so $\lambda x.MNP\langle y := Q \rangle$ is $\lambda x.((MN)(P\langle y := Q \rangle))$.

The λxgc -terms include as a subset the ordinary λ -terms, Λ ; we will say that a λxgc -term is 'pure' if it is also a λ -term, *i.e.*, if it has no subterms of the form $M\langle x := N \rangle$. The usual β -reduction is denoted $\overrightarrow{\beta}$

(and only defined on pure λxgc -terms, of course). Several other standard concepts generalise naturally to λxgc -terms:

Definitions 2.2 (λ -term concepts).

- The *free variable set* of a λxgc -term M is denoted $\text{fv}(M)$ and defined inductively over M by (the usual) $\text{fv}(x) = \{x\}$, $\text{fv}(\lambda x.M) = \text{fv}(M) \setminus \{x\}$, $\text{fv}(MN) = \text{fv}(M) \cup \text{fv}(N)$, and (the new) $\text{fv}(M\langle x := N \rangle) = (\text{fv}(M) \setminus \{x\}) \cup \text{fv}(N)$. A λx -term M is *closed* iff $\text{fv}(M) = \emptyset$; the closed λx -terms are denoted Λx° .
- The result of *renaming all free occurrences of y in M to z* is written $M[y := z]$. It is as for λ -calculus plus (the new) $(M\langle x := N \rangle)[y := z] = M[x := x'] [y := z] \langle x' := N[y := z] \rangle$ with $x' \notin \text{fv}(\lambda x.M) \cup \{y, z\}$.
- Two terms are α -*equivalent*, written $M \equiv N$, is as for λ -calculus plus (the new) $M\langle x := N \rangle \equiv P\langle y := Q \rangle$ if $N \equiv Q$ and $M[x := z] \equiv P[y := z]$ for $z \notin \text{fv}(MP)$.
- The substitution $\langle x := N \rangle$ in $M\langle x := N \rangle$ is called *garbage* if $x \notin \text{fv}(M)$.

Remark 2.3. One might argue that

$$\text{fv}(M\langle x := N \rangle) = \begin{cases} \text{fv}(M) & \text{if } x \notin \text{fv}(M), \\ (\text{fv}(M) \setminus \{x\}) \cup \text{fv}(N) & \text{if } x \in \text{fv}(M) \end{cases} \quad (*)$$

is a better definition since when $x \notin \text{fv}(M)$ it is impossible for a substitution $\langle y := P \rangle$ in $M\langle x := N \rangle\langle y := P \rangle$ to substitute P for any variable in N . We will not do this, however, since it would interfere with garbage collection in an unnatural way, for example it would allow the reduction $x\langle y := z \rangle\langle z := v \rangle\langle z := w \rangle \xrightarrow{\text{gc}} x\langle y := z \rangle\langle z := w \rangle$ which seems to change the binding of the inner z .

Also note that $M\langle x := N \rangle$ has the same free variables as $(\lambda x.M)N$; in these terms free occurrences of x in M are bound by $_ \langle x := N \rangle$ respectively $\lambda x._$. Using $(*)$ would mean that contraction of a redex ((b) below) could remove free variables from a term which seems undesirable.

We will in general work modulo α -equivalence, using \equiv for identity of terms. In order to avoid trivial conflicts of variable names we will use the usual convention of Barendregt (1984, 2.1.13).

Convention 2.4 (bound variable naming). When a group of terms occurs in a mathematical context (like a definition, proof, etc.), then all variables bound in these terms are chosen to be distinct and different from all free variables in them.

Now we are ready to define ‘raw’ λxgc -reduction:

Definitions 2.5 (‘raw’ reduction step).

- Substitution generation*, $\xrightarrow{\text{b}}$, is the contextual closure (modulo \equiv) of

$$(\lambda x.M)N \rightarrow M\langle x := N \rangle \quad (\text{b})$$

- Explicit substitution*, $\xrightarrow{\text{x}}$, is defined as the contextual closure (modulo \equiv) of the union of

$$x\langle x := N \rangle \rightarrow N \quad (\text{xv})$$

$$x\langle y := N \rangle \rightarrow x \quad \text{if } x \neq y \quad (\text{xvgc})$$

$$(\lambda x.M)\langle y := N \rangle \rightarrow \lambda x.M\langle y := N \rangle \quad (\text{xab})$$

$$(M_1 M_2)\langle y := N \rangle \rightarrow M_1\langle y := N \rangle M_2\langle y := N \rangle \quad (\text{xap})$$

Note that by 2.4 in (xab), $x \neq y$ and $x \notin \text{fv}(N)$, so there is neither variable capture nor clash.

- Garbage collection* $\xrightarrow{\text{gc}}$ is the contextual closure (modulo \equiv) of

$$M\langle x := N \rangle \rightarrow M \quad \text{if } x \notin \text{fv}(M) \quad (\text{gc})$$

- $\xrightarrow{\text{xgc}}$ is the union $\xrightarrow{\text{x}} \cup \xrightarrow{\text{gc}}$.

‘Raw’ λ xgc-reduction, $\xrightarrow{\text{bxgc}}$, is the union $\xrightarrow{\beta} \cup \xrightarrow{\text{xgc}}$ (we will generally use concatenation of subscripts to denote the union of relations).

Remark 2.6. The above reduction definition is based directly on the usual definitions of β -reduction by substitution by writing it explicitly, *i.e.*, changing the implicit definition of ‘meta-substitution’ to an explicit syntactic substitution without changing anything else. In addition we have just added the definition of garbage collection directly: this is why we call it ‘raw’ reduction in contrast to the garbage-free reduction to be introduced in section 3 (where garbage collection is automatic).

Remark 2.7. As might be expected, it takes more reduction steps to reduce a term to normalform using $\xrightarrow{\text{bxgc}}$ than it takes using $\xrightarrow{\beta}$. To illustrate this, we mention that the longest $\xrightarrow{\text{bxgc}}$ -reduction path of the term $\lambda v.(\lambda x.I(Ix))(Iv)$ consists of 18 steps, the shortest of 8 whereas all the $\xrightarrow{\beta}$ -reduction paths of it consist of 4 steps. For the term $(\lambda fx.f(fx))(\lambda fx.f(fx))$ these numbers are 148, 33 and 5 or 6 respectively.

Next we establish properties of the subrelations that are essential when relating the raw calculus to the pure λ -calculus, in particular when explaining the relationship between explicit substitution, garbage collection, and traditional (implicit) substitution.

Proposition 2.8. $\xrightarrow{\beta} \diamond$ and $\xrightarrow{\beta} \text{SN}$.

Proposition 2.9. $\xrightarrow{\text{xgc}} \text{SN}$ and $\xrightarrow{\text{xgc}} \text{CR}$.

Proof. Strong normalisation is shown by finding a map $h : \Lambda x \rightarrow \mathbb{N}$ such that for all $M \xrightarrow{x} N$: $h(M) > h(N)$. The map defined inductively by $h(x) = 1$, $h(MN) = h(M) + h(N) + 1$, $h(\lambda x.M) = h(M) + 1$, and $h(M\langle x := N \rangle) = h(M) \times (h(N) + 1)$, is easily checked to satisfy this.

Local confluence follows from the observation that the three critical pairs can be easily solved. \square

The same argument can be applied to the component relations \xrightarrow{x} and $\xrightarrow{\text{gc}}$ without any change:

Propositions 2.10. a. $\xrightarrow{x} \text{SN}$ and $\xrightarrow{x} \text{CR}$. b. $\xrightarrow{\text{gc}} \text{SN}$ and $\xrightarrow{\text{gc}} \text{CR}$.

We will introduce abbreviations for the normal forms of these relations since we have now established that they are unique and will use them frequently below. *Note:* \multimap is defined as ‘the restriction of \multimap to reductions to normal form’ (thus $\multimap \subseteq \multimap$).

Definitions 2.11.

- a. $\downarrow_x(M)$ is the \xrightarrow{x} -nf of M , *i.e.*, $M \xrightarrow{x} \downarrow_x(M)$; M is *pure* if $M \equiv \downarrow_x(M)$.
- b. $\downarrow_{\text{gc}}(M)$ is the $\xrightarrow{\text{gc}}$ -nf of M , *i.e.*, $M \xrightarrow{\text{gc}} \downarrow_{\text{gc}}(M)$; M is *garbage-free* if $M \equiv \downarrow_{\text{gc}}(M)$.

Clearly this notion of a ‘pure’ term corresponds to the informal use of the concept above as ‘ordinary λ -term’.

With this we are able to establish the relation between explicit and pure substitution: we show that the behaviour of the substitutions $P\langle x := Q \rangle$ relates well to the usual meta-substitution $P[x := Q]$.

Lemma 2.12 (representation). For all terms M, N and variable x ,

$$\downarrow_x(M\langle x := N \rangle) \equiv \downarrow_x(M)[x := \downarrow_x(N)]$$

where $P[x := Q]$ denotes the term obtained by (usual) substitution of the (pure) term Q for all free occurrences of x in (the pure term) P as a meta-operation.

Proof. We show by induction on the number of symbols in M, N_1, \dots, N_n that

$$\downarrow_x(M\langle x_1 := N_1 \rangle \cdots \langle x_n := N_n \rangle) \equiv \downarrow_x(M)[x_1 := \downarrow_x(N_1)] \cdots [x_n := \downarrow_x(N_n)]$$

\square

A consequence of this is that for pure M and N , $\downarrow_x(M\langle x := N \rangle) \equiv M[x := N]$ as we should expect. Another obvious yet interesting consequence is the following:

Corollary 2.13 (substitution lemma).

$$M\langle x := N \rangle \langle y := P \rangle \xrightarrow{\text{xgc}} M\langle y := P \rangle \langle x := N \rangle \langle y := P \rangle$$

In the (non-explicit) λ -calculus these are identical terms; we'll discuss the consequences of this difference for preservation of strong normalisation in the next section.

Next several useful results, including the relationship of $\xrightarrow{\text{x}}$ with garbage collection.

Propositions 2.14. For any M ,

- a. If $M \xrightarrow{\text{gc}} N$ then $\text{fv}(M) \supseteq \text{fv}(N)$ and $\downarrow_x(M) \equiv \downarrow_x(N)$.
- b. If $M \xrightarrow{\text{x}} N$ then $\text{fv}(M) \supseteq \text{fv}(N)$.
- c. If $M \xrightarrow{\text{b}} N$ then $\text{fv}(M) = \text{fv}(N)$.
- d. If M is garbage-free then $\text{fv}(M) = \text{fv}(\downarrow_x(M))$.

A consequence of 2.14a is that we do not need to introduce a special notation for $\xrightarrow{\text{xgc}}$ -normal forms since any pure term is also garbage-free, *i.e.*, $\xrightarrow{\text{xgc}} = \xrightarrow{\text{x}}$.

Remark 2.15. Considering the above, one might ask why $\xrightarrow{\text{gc}}$ is needed at all when its effect seems obtainable by $\xrightarrow{\text{x}}$? The answer is that its effect is not obtainable by $\xrightarrow{\text{x}}$: intuitively $\xrightarrow{\text{gc}}$ does a single global garbage collection whereas $\xrightarrow{\text{x}}$ moves a substitution into a term, working a bit of the way towards garbage collection. Formally, not only is $\xrightarrow{\text{x}} \not\subseteq \xrightarrow{\text{gc}}$ but also $\xrightarrow{\text{gc}} \not\subseteq \xrightarrow{\text{x}}$: the second inclusion fails because a single $\xrightarrow{\text{gc}}$ can remove a single substitution which can't be removed by $\xrightarrow{\text{x}}$ when it is not the innermost, *e.g.*, $x\langle x := y \rangle \langle v := w \rangle \xrightarrow{\text{gc}} x\langle x := y \rangle$ which can't be done by $\xrightarrow{\text{x}}$ (also see remark 2.3).

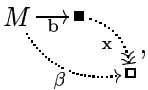
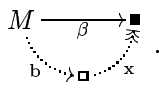
Furthermore, $\xrightarrow{\text{gc}}$ permits to reduce more efficient using clever strategies. One of the problems with explicit substitution is that there is a tendency to create substitutions that do not bind a variable, as is the case in $(x(fyyy))\langle x := I \rangle \xrightarrow{\text{x}} x\langle x := I \rangle(fyyy)\langle x := I \rangle$

Without $\xrightarrow{\text{gc}}$, the reduction to normal form of the latter term will involve distributing $\langle x := I \rangle$ three times over $fyyy$ and then deleting it four times using $\xrightarrow{\text{xvge}}$. $\xrightarrow{\text{gc}}$ allows to throw away the useless substitution $\langle x := I \rangle$ in $(fyyy)\langle x := I \rangle$ right after its creation, and hence $fyyy$ can be reached more efficient with respect to the length of the reduction path as well as with respect to the size of the terms along the reduction path.

The infinite reduction of the counterexample of Melliès (1995) consists mainly of reductions inside these useless substitutions. The need to avoid these useless substitutions has already been discerned by Kennaway and Sleep (1988) who chose not to create them at all, at the cost of restricting the evaluation order of nested β -redexes. In the calculus λxgc there are no restrictions on the evaluation order and still garbage in terms can be avoided.

Another reason for having the reduction $\xrightarrow{\text{gc}}$ is that it allows to attack the question whether $\xrightarrow{\text{bxgc}}$ has the PSN-property in a modular way. First we can study reductions where all garbage is removed as soon as possible. This has the advantage that no reductions take place inside garbage which will allow us to prove PSN by projection (cf. 2.17b, 3.5). Using the intermediate result we can prove PSN for the unrestricted reduction by carefully analyzing what can happen inside garbage.

Now that we have established the properties of the substitution subrelation, we can compare to the ordinary (pure) β -reduction. We show that $\xrightarrow{\text{bxgc}}$ is a conservative extension of $\xrightarrow{\beta}$, and thus confluent. We first relate single reduction steps and finally full reductions.

Proposition 2.16. For pure M , a. , and b. .

Proof. Follows from the observation that we can choose to contract the same β -redex with $\xrightarrow{\beta}$ and $\xrightarrow{\text{b}}$, and that the result of the $\xrightarrow{\text{b}}$ -reduction will have exactly one substitution in it. We then use uniqueness of $\xrightarrow{\text{x}}$ -normalforms and 2.12. \square

Below, 2.16a is generalised to non-pure terms because we will need this in the confluence proof below; in fact we strengthen it slightly to the special case where the contracted term is garbage-free (*cf.* 2.11b) since this will be needed for the proof of preservation of strong normalisation in the following section.

Lemmas 2.17 (projection). a. For all M , $\begin{array}{ccc} M & \xrightarrow{b} & N \\ \downarrow x & & \downarrow x \\ \downarrow_x(M) & \xrightarrow{\beta} & \downarrow_x(N) \end{array}$.

b. For garbage-free M , $\begin{array}{ccc} M & \xrightarrow{b} & N \\ \downarrow x & & \downarrow x \\ \downarrow_x(M) & \xrightarrow{\beta^+} & \downarrow_x(N) \end{array}$.

With this we are able to prove that $\xrightarrow{\text{bxgc}}$ is a conservative extension of $\xrightarrow{\beta}$:

Theorem 2.18. For pure terms M, N : $M \xrightarrow{\text{bxgc}} N$ iff $M \xrightarrow{\beta} N$.

Proof. **Case \Leftarrow :** Assume $M \xrightarrow{\beta} N$; the case then follows by induction on the length of the $\xrightarrow{\beta}$ -reduction, using 2.16b in each step.

Case \Rightarrow : Assume $M \xrightarrow{\text{bxgc}} N$ and M, N pure. We will do induction on the length of the $\xrightarrow{\text{bxgc}}$ -reduction and prove

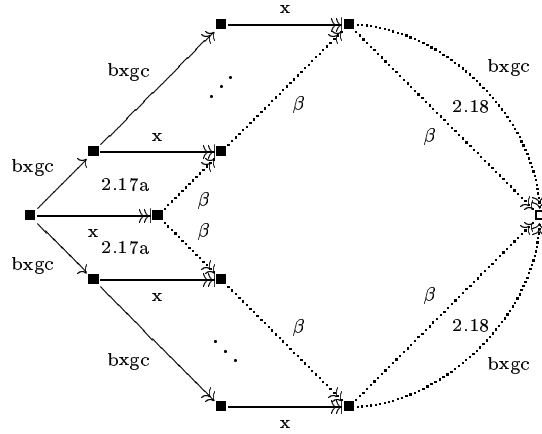
$$\begin{array}{ccccccc} M & \xrightarrow{\text{bxgc}} & M_1 & \xrightarrow{\text{bxgc}} & \dots & \xrightarrow{\text{bxgc}} & M_{n-1} & \xrightarrow{\text{bxgc}} & N \\ \parallel & & \downarrow x & & & & \downarrow x & & \parallel \\ M & \xrightarrow{\beta} & \downarrow_x(M_1) & \xrightarrow{\beta} & \dots & \xrightarrow{\beta} & \downarrow_x(M_{n-1}) & \xrightarrow{\beta} & N \end{array}$$

Each step in the top of each ‘cell’ is one of $\xrightarrow{\text{gc}}$, \xrightarrow{x} , and \xrightarrow{b} , thus each ‘cell’ is an instance of either 2.14a, confluence of \xrightarrow{x} , or 2.17a. \square

An easy consequence of all this is confluence of the raw calculus based on $\xrightarrow{\beta}\text{CR}$.

Corollary 2.19. $\xrightarrow{\text{bxgc}}\text{CR}$.

Proof. The following diagram exploits $\xrightarrow{\beta}\Diamond$ to show $\xrightarrow{\text{bxgc}}\Diamond$:



(The technique is called the ‘interpretation method’ by Hardin and Lévy (1990).) \square

It is easy to verify that all the results in this section hold for the non-garbage collection variant $\xrightarrow{\text{bx}}$ -reduction (where rule (gc) is omitted), however, in the following section (gc) will prove essential.

Remark 2.20. Finally we observe that the calculus λxgc does not have *confluence on open terms*, that is, the kind of ‘higher-order confluence’ defined as confluence on metaterms allowing metavariables for terms (the substitution lemma does not hold for instance). Whereas for a de Bruijn indices based calculus like $\lambda\sigma$ confluence on open terms is a useful property, for λxgc it doesn’t make sense. Consider for instance the term $A\langle x := y \rangle$ where A is a metavariable for terms. It is not clear whether $A\langle x := y \rangle \xrightarrow{\text{gc}} A$. Similar problems arise for abstractions over metavariables and α -conversion.

3 Preservation of Strong Normalisation

The presence of explicit substitutions in λxgc means that we can have terms with *garbage*, *i.e.*, with subterms $M\langle x := N \rangle$ where $x \notin \text{fv}(M)$, which means that there is still garbage to collect.

In this section we will first discuss briefly why it is important to investigate calculi of explicit substitution that are more abstract than $\lambda\sigma$ of Abadi et al. (1991) in order to study preservation of strong normalisation of λ -terms (PSN). We then introduce our main technical tool to achieve this goal, ‘garbage-free’ reduction where garbage collection is made implicit. We define it naively as a restriction of ‘raw’ reduction, and show that it is a confluent refinement of the λ -calculus with PSN. Afterwards we outline how the result can be generalised to show that the raw λxgc -calculus of section 2 also has PSN by showing that the ‘garbage collection overhead’ does not provide for infinite reductions. Our proof is direct, furthermore, by reasoning more carefully and avoiding infinite reduction sequences, the proof can be made constructive.

Remark 3.1. Why have we not considered a ‘stronger’ calculus with some kind of composition of substitutions like $\lambda\sigma$ (Abadi et al. 1991) has? The strength of such calculi lies in the fact that the substitution lemma is internalised whereas for simpler calculi like λxgc and λv (Lescanne 1994) it is a theorem. The weakness lies in the fact that the resulting more complex definition of substitution may give problems, *e.g.*, break PSN.

To be precise: for λxgc the substitution lemma 2.13 is a property of reduction. Of course it is dangerous to add a rewrite rule

$$M\langle x := N \rangle\langle y := P \rangle \rightarrow M\langle y := P \rangle\langle x := N\langle y := P \rangle \rangle$$

since this rule immediately yields infinite rewrite possibilities whenever it is applicable. But one might consider parallel substitutions $\langle x_1, \dots, x_m := N_1, \dots, N_m \rangle$, abbreviated by $\langle \vec{x} := \vec{N} \rangle$, which intuitively means substitute N_1 for x_1 , N_2 for x_2 , \dots , N_m for x_m simultaneously, and the following rewrite rule:

$$M\langle \vec{x} := \vec{N} \rangle\langle \vec{y} := \vec{P} \rangle \rightarrow' M\langle \vec{x}, \vec{y} := N_1\langle \vec{y} := \vec{P} \rangle, \dots, N_m\langle \vec{y} := \vec{P} \rangle, P_1, \dots, P_n \rangle$$

\rightarrow' seems to be a harmless reduction rule and the straightforward extension of λxgc to parallel substitutions extended with this composition rule is close to $\lambda\sigma$ of Abadi et al., as was shown by Kamareddine and Nederpelt (1993). In fact, $\lambda\sigma$ has rules that provide for interaction between substitutions similar to the above mentioned rule, namely the rules called *Clos* and *Map*. Now using this rule it is not difficult to show that PSN does not hold; we present a counterexample similar to that of (Melliès 1995).

Let a, b be terms and y, y' variables such that $y, y' \notin \text{fv}(ab)$. Define substitutions

$$S_0 \equiv \langle y := (\lambda y.a)b \rangle, \quad S_{n+1} \equiv \langle y := bS_n \rangle$$

and consider the following derivations (for simplicity we offend the variable convention but this is easily repaired):

$$\begin{aligned} & (\lambda y.(\lambda y'.a)((\lambda y.a)b))((\lambda y.a)b) \rightarrow a\langle y' := ((\lambda y.a)b) \rangle\langle y := (\lambda y.a)b \rangle \\ & \rightarrow' a\langle y', y := ((\lambda y.a)b)\langle y := (\lambda y.a)b \rangle, (\lambda y.a)b \rangle \\ & \rightarrow a\langle y', y := (\lambda y.a\langle y := (\lambda y.a)b \rangle)(b\langle y := (\lambda y.a)b \rangle), (\lambda y.a)b \rangle \\ & \equiv a\langle y', y := (\lambda y.aS_0)(bS_0), (\lambda y.a)b \rangle \\ & \rightarrow a\langle y', y := aS_0\langle y := bS_0 \rangle, (\lambda y.a)b \rangle \\ & \equiv a\langle y', y := aS_0S_1, (\lambda y.a)b \rangle, \\ & aS_0S_{m+1} \equiv a\langle y := (\lambda y.a)b \rangle\langle y := bS_m \rangle \equiv a\langle y' := (\lambda y.a)b \rangle\langle y := bS_m \rangle \\ & \rightarrow' a\langle y', y := ((\lambda y.a)b)\langle y := bS_m \rangle, bS_m \rangle \\ & \rightarrow a\langle y', y := (\lambda y.a\langle y := bS_m \rangle)(b\langle y := bS_m \rangle), bS_m \rangle \\ & \equiv a\langle y', y := (\lambda y.aS_{m+1})(bS_{m+1}), bS_m \rangle \\ & \rightarrow a\langle y', y := aS_{m+1}\langle y := bS_{m+1} \rangle, bS_m \rangle \\ & \equiv a\langle y, y' := aS_{m+1}S_{m+2}, bS_m \rangle, \end{aligned}$$

and $aS_{m+1}S_{n+1} \equiv a\langle y' := bS_m \rangle\langle y := bS_n \rangle \rightarrow' a\langle y', y := bS_m\langle y := bS_n \rangle, bS_n \rangle \equiv a\langle y', y := bS_mS_{n+1}, bS_m \rangle$

which combine into an infinite derivation in the following schematic way:

$$\begin{aligned} (\lambda y.(\lambda y'.a)((\lambda y.a)b))((\lambda y.a)b) &\rightarrow \cdots S_0 S_1 \cdots \rightarrow \cdots S_1 S_2 \cdots \rightarrow \cdots S_0 S_2 \cdots \\ &\rightarrow \cdots S_2 S_3 \cdots \rightarrow \cdots S_1 S_3 \cdots \rightarrow \cdots S_0 S_3 \cdots \rightarrow \cdots S_3 S_4 \cdots \quad \cdots \end{aligned}$$

Note that all the reductions (but the first three) take place inside garbage and that it is essential that substitutions can be shifted into garbage. In fact, even the rewrite rule

$$M\langle x := N \rangle \langle y := P \rangle \rightarrow'' M\langle x := N \langle y := P \rangle \rangle \quad \text{if } y \notin \text{fv}(M)$$

(no parallel substitutions are needed) corrupts PSN as can be shown in a similar way.

Considering these negative results on calculi with composition of substitutions, it seems important first to study a calculus of explicit substitutions without composition. Once we know that such a calculus has PSN, we can try to add restricted rules for composition of substitutions without losing PSN. We now proceed to show PSN for λxgc .

Definition 3.2 (garbage-free reduction). $\xrightarrow{\text{bx}\downarrow\text{gc}}$ is the union of compositions $(\xrightarrow{x} \cdot \xrightarrow{g}) \cup (\xrightarrow{b} \cdot \xrightarrow{g})$.

We will denote the garbage-free reduction calculus $\lambda x\downarrow gc$ (the modulo-symbol ‘/’ should here be read “composed with complete reduction to normal form by” so for example $\xrightarrow{\beta} = \xrightarrow{b/x}$ by 2.16).

We start by proving confluence. All the results here will depend heavily on the knowledge that each garbage-free reduction step is a single \xrightarrow{x} - or \xrightarrow{b} -step followed by \xrightarrow{g} -reduction to gc-nf.

Lemmas 3.3. a. For all terms M, N : if $M \xrightarrow{\text{bx}\downarrow\text{gc}} N$ then $\downarrow_{gc}(M) \xrightarrow{\text{bx}\downarrow\text{gc}} \downarrow_{gc}(N)$.

b. For garbage-free M ,

$$\begin{array}{ccc} M & \xrightarrow{\text{bx}\downarrow\text{gc}} & N \\ & \searrow \text{gc} & \downarrow \text{gc} \\ & \text{bx}\downarrow\text{gc} & \downarrow_{gc}(N) \end{array}$$

Theorem 3.4. $\xrightarrow{\text{bx}\downarrow\text{gc}} \text{CR}$.

Proof. We show $\xrightarrow{\text{bx}\downarrow\text{gc}} \Diamond$: it follows from $\xrightarrow{\text{bx}\downarrow\text{gc}} \subseteq \xrightarrow{\text{bx}\downarrow\text{gc}}$, the confluence of $\xrightarrow{\text{bx}\downarrow\text{gc}}$, and (two applications of) 3.3 by a simple diagram chase. \square

Theorem 3.5 (PSN for $\lambda x\downarrow gc$). *Pure terms that are $\xrightarrow{\beta}$ -strongly normalising are also strongly normalising for $\xrightarrow{\text{bx}\downarrow\text{gc}}$.*

Proof. We will show that each $\xrightarrow{\text{bx}\downarrow\text{gc}}$ -reduction corresponds to a $\xrightarrow{\beta}$ -reduction of comparable length, hence there are no infinite $\xrightarrow{\text{bx}\downarrow\text{gc}}$ -reductions for terms that are strongly normalising for $\xrightarrow{\beta}$. The proof is similar to that of 2.18.

Assume M is pure and strongly normalising for $\xrightarrow{\beta}$. Since M is pure it has no \xrightarrow{xgc} -redexes and every $\xrightarrow{\text{bx}\downarrow\text{gc}}$ -reduction starting with M , whether finite or not, is of the form $M \equiv M_0 \xrightarrow{b} M_1 \xrightarrow{xgc} M_2 \xrightarrow{b} M_3 \xrightarrow{xgc} \cdots$ where the “ \xrightarrow{xgc} ” segments are really $\xrightarrow{g} \cdot (\xrightarrow{x} \cdot \xrightarrow{g}) \cdots (\xrightarrow{x} \cdot \xrightarrow{g})$ sequences. From such a sequence we can construct a $\xrightarrow{\beta}$ -sequence as follows, from left to right:

$$\begin{array}{ccccccc} M \equiv M_0 & \xrightarrow{b} & M_1 & \xrightarrow{xgc} & M_2 & \xrightarrow{b} & M_3 \xrightarrow{xgc} \cdots \\ \parallel & & \downarrow x & & \downarrow x & & \downarrow x \\ & \text{2.17b} & & \text{2.14a} & & \text{2.17b} & \text{2.14a} \\ \downarrow_x(M_0) & \cdots \xrightarrow{\beta} & \downarrow_x(M_1) & \equiv & \downarrow_x(M_2) & \cdots \xrightarrow{\beta} & \downarrow_x(M_3) \equiv \cdots \end{array}$$

Since the lower reduction is finite and $\xrightarrow{xgc} \text{SN}$, the upper one must also be finite. \square

Now we are ready to make garbage collection explicit again. In order to distinguish between the degrees of ‘garbage reductions’ such that we can identify the garbage collection overhead, we define two mutually disjoint classes of reductions: garbage-reductions and reductions outside garbage.

Definitions 3.6. We subdivide the reduction relation $\xrightarrow{\text{bx}\downarrow\text{gc}}$ into two mutually disjoint parts.

a. *Garbage-reduction* is the contextual closure of the reduction generated by:

- If $N \xrightarrow{\text{bxgc}} N'$ and $x \notin \text{fv}(\downarrow_{\text{gc}}(M))$ then $M\langle x := N \rangle \xrightarrow{\text{bxgc}} M\langle x := N' \rangle$ is garbage-reduction.
- if $x \notin \text{fv}(\downarrow_{\text{gc}}(MN))$ then $(MN)\langle x := P \rangle \xrightarrow{\text{bxgc}} (M\langle x := P \rangle)(N\langle x := P \rangle)$ is garbage-reduction,
- if $x \notin \text{fv}(\downarrow_{\text{gc}}(\lambda y.M))$ then $(\lambda y.M)\langle x := N \rangle \xrightarrow{\text{bxgc}} \lambda y.M\langle x := N \rangle$ is garbage-reduction,
- if $x \notin \text{fv}(M)$ then $M\langle x := N \rangle \xrightarrow{\text{bxgc}} M$ is garbage-reduction,

Note the use of $\text{fv}(\downarrow_{\text{gc}}(\cdot))$ to ensure that for instance $(x\langle y := z \rangle x)\langle z := M \rangle \xrightarrow{x} (x\langle y := z \rangle \langle z := M \rangle)(x\langle z := M \rangle)$ is garbage-reduction.

- b. Reduction *outside garbage* is any reduction that is not garbage-reduction; this is equivalent to saying that the contracted redex has no descendant in the $\xrightarrow{\text{gc}}$ -normalform.

With this we can prove the following by induction on the structure of terms.

Proposition 3.7.

- a. If $M \xrightarrow{\text{bxgc}} N$ is garbage-reduction then $\downarrow_{\text{gc}}(M) \equiv \downarrow_{\text{gc}}(N)$.
- b. If $M \xrightarrow{\text{bxgc}} N$ is outside garbage then $\downarrow_{\text{gc}}(M) \xrightarrow{\text{bxl}_{\text{gc}}} \downarrow_{\text{gc}}(N)$.

Definition 3.8. We say N is *body of a substitution in M* if for some P, x , $P\langle x := N \rangle$ is a subterm of M . The predicate $\text{subSN}(M)$ should be read to be *all bodies of substitutions in M are strongly normalising for $\xrightarrow{\text{bxgc}}$ -reduction*.

The following lemma expresses our intuition about garbage-reduction.

- Lemma 3.9.**
- a. If $\text{subSN}(M)$ and $M \xrightarrow{\text{bxgc}} N$ is garbage-reduction, then $\text{subSN}(N)$.
 - b. If $\text{subSN}(M)$ then M is strongly normalising for garbage-reduction.

Definition 3.10. Define for all terms M , $\# \text{gf}(M)$ to be the maximum length of garbage-free reduction paths starting in $\downarrow_{\text{gc}}(M)$. Note that $\# \text{gf}(M)$ can be infinite as it is for $(\lambda x.xx)(\lambda x.xx)$.

Theorem 3.11. If $\# \text{gf}(M) < \infty$ and $\text{subSN}(M)$ then M is strongly normalising for $\xrightarrow{\text{bxgc}}$ -reduction.

Proof sketch. We use induction on $\# \text{gf}(M)$ using an extra induction over the term structure to show that the $\text{subSN}(\cdot)$ property is preserved for any reduct (this requires 3.9b, 3.7, and 3.9a). \square

Corollary 3.12 (PSN for λxgc). A pure term is strongly normalising for β -reduction if and only if it is strongly normalising for $\xrightarrow{\text{bxgc}}$ -reduction.

Proof. **Case \Rightarrow .** If M is pure then $\text{subSN}(M)$ and if M is strongly normalising for $\xrightarrow{\beta}$ -reduction then by 3.5, $\# \text{gf}(M) < \infty$. Now use 3.11.

Case \Leftarrow . By 2.16b infinite $\xrightarrow{\beta}$ -reductions induce infinite $\xrightarrow{\text{bxgc}}$ -reductions. \square

The following corollary characterises which arbitrary terms of Λx are SN for $\xrightarrow{\text{bxgc}}$ -reduction.

Corollary 3.13. A λxgc -term M is SN for $\xrightarrow{\text{bxgc}}$ -reduction if and only if for all subterms N of M : $\downarrow_x(N)$ is SN for $\xrightarrow{\beta}$.

Proof. The only if part is easy. We prove the if part by induction on the maximal number of nestings of substitutions in M . If the maximal number of nestings is 0 then use 3.12.

Suppose that the maximal number of nestings of substitutions is $n + 1$. By the induction hypothesis, for all bodies N of substitutions in M : N is SN for $\xrightarrow{\text{bxgc}}$ -reduction; therefore, $\text{subSN}(M)$. After one garbage-free reduction step $M \xrightarrow{\text{bxl}_{\text{gc}}} M'$, every garbage-free reduction path of M' will correspond to a $\xrightarrow{\beta}$ -reduction path of $\downarrow_x(M)$ similar to the proof of 3.5. Therefore, $\# \text{gf}(M) < \infty$. Now by 3.11, M is SN for $\xrightarrow{\text{bxgc}}$ -reduction. \square

Remark 3.14. Finally we mention a positive result: adding the rewrite rule

$$M\langle x := N \rangle \langle y := P \rangle \rightarrow''' M\langle x := N \rangle \langle y := P \rangle \quad \text{if } x \in \text{fv}(\downarrow_{\text{gc}}(M)) \text{ and } y \notin \text{fv}(M)$$

does not break PSN; see Bloo and Geuvers (1995) for details. Intuitively the reason for this is that if $x \in \text{fv}(\downarrow_{\text{gc}}(M))$ then in some $\xrightarrow{\text{bxgc}}$ -reduct of $M\langle x := N \rangle \langle y := P \rangle$, indeed a subterm $N\langle y := P \rangle$ will occur, and accelerating the occurrence of this subterm can do no harm. In the rewrite rules discussed in remark 3.1 substitutions are being introduced that cannot occur in a \rightarrow reduction path of $M\langle x := N \rangle \langle y := P \rangle$ and this is what breaks PSN, since then an infinite loop can be generated by composing substitutions with a redex inside and distributing over the redex.

4 Conclusions

We have introduced λxgc , a new calculus of explicit substitutions which is in a sense more ‘pure’ than other explicit substitution calculi because *only* substitution (not renaming) is made explicit. We have shown that it is confluent and preserves strong normalisation. Our proof is less involved than the proofs of PSN for similar calculi known to us. Furthermore we have argued that the reason for $\lambda\sigma$ not having the PSN property is in fact that in $\lambda\sigma$ substitutions can be shifted into garbage where they can multiply, and we have shown that extending λxgc with reductions that have such an effect breaks PSN.

Further work. Extensions to this work proceed in several directions. First, we are generalising the results to combinatory reduction systems (Rose 1995). Second, we are working on how to model sharing of identical subterms within explicit substitution and deriving efficient abstract reduction machines from them, this includes generalising the results to ‘explicit cyclic substitution’ of Rose (1993). A third direction is defining typing rules for the terms of λxgc to get a typed λ -calculus of explicit substitutions which is strongly normalising.

Acknowledgements. We would like to thank Daniel Briaud, Herman Geuvers, Neil Jones, Pierre Lescanne, Rob Nederpelt, Alejandro Ríos, Eva Rose, and the anonymous referees for fruitful discussions and comments.

References

- Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J.-J. (1991). Explicit substitutions. *Journal of Functional Programming* 1(4): 375–416.
- Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics*. Revised edn. North-Holland.
- Benaissa, Z.-E.-A., Briaud, D., Lescanne, P. and Rouyer-Degli, J. (1995). λv , a calculus of explicit substitutions which preserves strong normalisation. *Rapport de Recherche 2477*. INRIA, Lorraine. Technôpole de Nancy-Brabois, Campus Scientifique, 615 rue de Jardin Botanique, BP 101, F-54600 Villers lès Nancy. (URL: <http://www.loria.fr/~lescanne/PUBLICATIONS/RR-2477.PS>)
- Bloo, R. (1995). Preservation of strong normalisation for explicit substitution. *Technical report*. Eindhoven University of Technology, Dept. of Computer Science. P.O.box 513, 5600 MB Eindhoven, The Netherlands. TUE report 95-08.
- Bloo, R. and Geuvers, H. (1995). Preservation of strong normalisation for some explicit substitution calculi. *Technical report*. Eindhoven University of Technology, Dept. of Computer Science. P.O.box 513, 5600 MB Eindhoven, The Netherlands. Presented at (?), in preparation as TUE report.
- Crégut, P. (1990). An abstract machine for the normalisation of λ -terms. *1990 ACM Conference on LISP and Functional Programming*. Nice, France. pp. 333–340.
- Curien, P.-L. (1986). Categorical combinators. *Information and Control* 69: 188–254.
- Curien, P.-L. (1990). An abstract framework for environment machines. Unpublished note from LIENS/CNRS.
- Hardin, T. and Lévy, J.-J. (1990). A confluent calculus of substitutions. *Rapport de Recherche 90-11*. CEDRIC. 292, rue Saint-Martin, 75141 Paris CEDEX 03. Also in France-Japan Artificial Intelligence and Computer Science Symposium, Izu, 1989.
- Huet, G. (1980). Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM* 27(4): 797–821.

- Kamareddine, F. and Nederpelt, R. (1993). On stepwise explicit substitution. *International Journal of Foundations of Computer Science* 4(3): 197–240.
- Kamareddine, F. and Ríos, A. (1995). A λ -calculus à la de Bruijn with explicit substitutions. Personal communication.
- Kennaway, J. R. and Sleep, R. (1988). Director strings as combinators. *ACM Transactions on Programming Languages and Systems* 10: 602–626.
- Klop, J. W. (1987). Term rewriting systems: a tutorial. *Bulletin of the European Association for Theoretical Computer Science* 32: 143–182.
- Landin, P. (1964). The mechanical evaluation of expressions. *Computer Journal* 6: 308–320.
- Lescanne, P. (1994). From $\lambda\sigma$ to λv : a journey through calculi of explicit substitutions. *POPL '94—21st Annual ACM Symposium on Principles of Programming Languages*. Portland, Oregon. pp. 60–69.
- Lescanne, P. and Rouyer-Degli, J. (1995). Explicit substitutions with de Bruijn's levels. In P. Lescanne (ed.), *Rewriting Techniques and Applications 1995*. Kaiserslautern, Germany.
- Melliès, P.-A. (1995). Typed λ -calculi with explicit substitution may not terminate. In M. Dezani (ed.), *Int. Conf. on Typed Lambda Calculus and Applications*. LNCS. U of Edinburgh. Springer-Verlag. Edinburgh, Scotland.
- Ríos, A. (1993). *Contributions à l'étude des λ -calculs avec des substitutions explicites*. Thèse de doctorat. U Paris VII.
- Rose, K. H. (1993). Explicit cyclic substitutions. *Semantics Note D-166*. DIKU (University of Copenhagen). Universitetsparken 1, DK-2100 København Ø, Denmark. [URL: ftp://ftp.diku.dk/diku/semantics/papers/D-166.ps](ftp://ftp.diku.dk/diku/semantics/papers/D-166.ps)
- Rose, K. H. (1995). Combinatory reduction systems with explicit substitution for computation. In B. Möller (ed.), *HOA '95 – Second International Workshop on Higher-Order Algebra, Logic and Term Rewriting*. Universität-GH Paderborn. [URL: http://www.diku.dk/~kris/research.html](http://www.diku.dk/~kris/research.html)
- Rosen, B. K. (1973). Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM* 20(1): 160–187.
- Turner, D. A. (1979). A new implementation technique for applicative languages. *Software Practice and Experience* 9: 31–49.