Combinatory Reduction Systems with Explicit Substitution that Preserve Strong Normalisation

Roel Bloo
Eindhoven University of Technology[*]
bloo@win.tue.nl

Kristoffer H. Rose
DIKU, University of Copenhagen[†]
kris@diku.dk

Submission for RTA '96, January 1996

Abstract

In this paper, we generalise the notion of explicit substitution from the λ-calculus to *higher order rewriting*, realised by combinatory reduction systems (CRSs). For every confluent CRS, R, we construct an explicit substitution variant, Rx, which we prove confluent.

We identify a large subset of the CRSs, the *structure-preserving CRSs*, and show for any structure-preserving CRS R that Rx preserves strong normalisation of R.

We believe that this is a significant first step towards providing a methodology for reasoning about the operational properties of higher-order rewriting in general, and higher-order program transformations in particular, since confluence ensures correctness of such transformations and preservation of strong normalisation ensures that the transformations are always safe, in both cases *independently of the used reduction strategy*.

Key words: combinatory reduction systems, explicit substitution, confluence, preservation of strong normalization.

Contents: 1. Introduction, 2. Combinatory Reduction Systems (CRS), 3. Explicit Substitutes, 4. Preservation of Strong Normalisation, 5. Conclusions, References.

[*]PO-Box 513, 5600 MB Eindhoven, The Netherlands; *tel: +31 402475004*
[†]Universitetsparken 1, DK–2100 København Ø, Denmark; *http://www.diku.dk/~kris/.*

1 Introduction

The λ-calculus is a useful basis for denotational models of the computable functions. However, counting β-reduction steps $(\lambda x.M)N \longrightarrow M[x := N]$ does not include the size of the abstraction body M. This means that it is not a useful operational model of computation because the number of β-reduction steps required to model a computation does not correspond to the 'work' a computer has to do.

The traditional solution to this problem has been to assume a fixed reduction strategy, and then invent abstract machines realising it in an operationally realistic way (Landin 1964, Henderson 1980). This is unsatisfying because it makes reasoning about the relative operational behaviour of reduction strategies difficult, *e.g.*, for showing the correctness of program transformations. A better solution in this respect is the study of explicit (or stepwise) substitution (Abadi, Cardelli, Curien and Lévy 1991) where each reduction step corresponds to a 'realistic' computation step without sacrificing the freedom to choose the reduction strategy.

Quite independent of this, Klop (1980) developed *combinatory reduction systems* (CRS) to provide a syntactic model for more general reduction systems with binding than pure λ-calculus. It seems natural to make this computational in the same sense as the λ-calculus by making explicit the steps involved in substitution. There is a technical problem, though: the explicit substitution tradition quoted above is based on λ-calculus with indices à la de Bruijn (1972) but CRS uses 'real' variables in an essential way.

Hence in this paper we study how the *named explicit substitution* principle of Rose (1992) can be applied to CRS, by systematising the way this was done for the λxgc-calculus of Rose and Bloo (1995), which we reproduce as a case study. We assume familiarity with the λ-calculus (Barendregt 1984).

2 Combinatory Reduction Systems

Combinatory reduction systems (CRSs) form an expressive class of reduction systems on the class of inductively defined terms extended with the variable binding notion of λ-calculus. We study CRS in this paper because they constitute a generalisation of the rewriting done by functional programming 'evaluation steps'.

The CRS formalism was invented by Klop (1980) for a systematic treatment of combinations of term rewrite systems (TRS) with the λ-calculus, inspired by the 'definable extensions of the λ-calculus' of Aczel (1978). However, CRS can also be understood as a form of 'higher order rewriting' (Nipkow 1991, van Oostrom and van Raamsdonk 1995).

This section gives a brief introduction to CRS. The presentation follows the (highly recommended) survey of Klop, van Oostrom and van Raamsdonk (1993) closely, with only a slight change of notation to facilitate induction over syntax.

2.1 Definition (combinatory reduction systems).

A. A set of *function symbols*, called the *alphabet*, and denoted $F, G, \ldots$; each annotated with a *fixed arity* (notation F^n).

B. The corresponding *preterms*, denoted $t, s, \ldots$, have the inductive form

$$t ::= x \mid [x]t \mid F^n(t_1, \ldots, t_n) \mid Z^n(t_1, \ldots, t_n)$$

where $x, y, z, \ldots$ are used to denote *variables* and X, Y, Z to denote *metavariables*, each with an explicit arity superscript as indicated[1]. The four different preterm forms are called *variable*, *metaabstraction*, *construction*, and *metaapplication*, respectively.

C. We denote the *set of metavariables* occuring in a preterm t as $\mathrm{mv}(t)$.

D. A preterm is *closed* if $\mathrm{fv}(t) = \varnothing$ where $\mathrm{fv}(x) = \{x\}$, $\mathrm{fv}([x].t) = \mathrm{fv}(t) \setminus \{x\}$, and $\mathrm{fv}(F^n(t_1, \ldots, t_n)) = \mathrm{fv}(Z^n(t_1, \ldots, t_n)) = \mathrm{fv}(t_1) \cup \cdots \cup \mathrm{fv}(t_n)$.

E. Metaterms are preterms considered *modulo renaming of (bound) variables*: we take as implicit the usual α-equivalence denoted $\equiv$ and defined inductively by $x \equiv x$, $[x]t \equiv [y]s$ if $t[x := z] \equiv s[y := z]$ for $z \notin \mathrm{fv}(s) \cup \mathrm{fv}(t)$, and $F^n(s_1, \ldots, s_n) \equiv F^n(t_1, \ldots, t_n)$ as well as $Z^n(s_1, \ldots, s_n) \equiv Z^n(t_1, \ldots, t_n)$ if $s_i \equiv t_i$ for each i; the renaming $t[y := z]$ is defined the usual way by $x[y := z] = z$ if $x = y$, $x[y := z] = x$ if $x \neq y$, $([x]t)[y := z] = [x']t[x := x'][y := z]$ with $x' \notin \mathrm{fv}([x]t) \cup \{y, z\}$, $F^n(t_1, \ldots, t_n)[y := z] = F^n(t_1[y := z], \ldots, t_n[y := z])$, and $Z^n(t_1, \ldots, t_n)[y := z] = Z^n(t_1[y := z], \ldots, t_n[y := z])$.

F. A *term* is a metaterm without metavariables (but possibly including metaabstraction which is then called *abstraction*).

G. *Rewrite rules*, written $p \to t$, have the following restrictions:

- the LHS (left hand side) p must be a *pattern*: it should be a construction, that is, there must be a function symbol at the root of p, furthermore p should be closed and the arguments of metaapplications should be distinct variables.
- the RHS (right hand side) t must be a *contractor* of the metavariables of the LHS: only those metavariables may occur.

H. A CRS is a set of rewrite rules over metaterms.

2.2 Notation (syntactic restrictions). We will sometimes define the preterms of a CRS by an inductive definition such as

$$a \ ::= \ x \mid L(a) \mid A(B(a), b)$$

From such a definition it is easily seen that the alphabet of the defined CRS is $\{L^1, A^2, B^1\}$. However, more importantly, it can be seen that B *will only occur as the first component of an* A*-construction.* We will use such *restricted (pre-/meta-)terms* freely when it is clear that the defined subsystem is closed with respect to reduction.

2.3 Notation (syntax and alphabet). We will allow 'implicit extraction' of an alphabet from an inductive definition of syntax in the obvious way, *i.e.*, a declaration such as

$$a \ ::= \ x \mid L(a) \mid A(a, b)$$

implies that the alphabet is L^1 and A^2.

[1]Thus for each particular alphabet the definition of preterms is finite and inductive.

2.4 Remark (term rewriting systems). Any term rewrite system (TRS) is also the closed part of a CRS containing no metaabstractions and only 0-ary metaapplications (namely the TRS "variables"). The restriction of the CRS relation to closed terms is the same as defined by the TRS.

2.5 Example ($\lambda\beta$-calculus). The ordinary untyped $\lambda\beta$-calculus is described by a subsystem of the CRS with function symbols $\{\lambda^1, @^2\}$ and the single rule

$$@^2(\lambda^1([x]Z^1(x)), Y^0()) \to Z^1(Y^0()) \qquad (\beta)$$

The λ-term $(\lambda x.yx)y$ corresponds to the CRS term $@(\lambda([x]@(y,x)), y)$ and reduces by (β) to $@(y,y)$ if $Z^1(x)$ is matched with $@(y,x)$.

The term $Z^1(Y^0())$ on the righthand side of (β) corresponds to the usual definition by substitution in that the bound variable to be substituted, x, is represented implicitly on the right side through the use of the metavariable Z^1 – informally we could have written $Z^1(Y^0())$ with ordinary substitution as something like $(Z^1(x))[x := Y^0()]$.

To represent untyped λ-terms this CRS is a bit too large; we must restrict to the terms defined by

$$t \ ::= \ x \ | \ \lambda([x]t) \ | \ @(t_1, t_2)$$

to get rid of terms like $[x]x$ and λx which do not correspond to λ-terms. Note that reduction of a restricted term yields another restricted term.

2.6 Notation (CRS abbreviations). We will use the usual CRS abbreviations:

- the arity superscript of function symbols and metavariables is omitted when obvious, and we omit the () after zero-ary symbols,
- $[x,y,z]t$ abbreviates the nested metaabstraction $[x][y][z]t$,
- $Fxyz.t$ abbreviates 'curried F-abstraction' $F([x](F([y](F([z]t)))))$,
- st abbreviates 'application' $@(s,t)$ (when it has no other interpretation) where $@^2$ is then included as a function symbol, and
- $\vec{x}_{(n)}$, $\vec{Z}_{(n)}$, and $\vec{t}_{(n)}$, abbreviate the sequences $x_1, \ldots, x_n$, $Z_1, \ldots, Z_n$, and $t_1, \ldots, t_n$, respectively (we omit the $\cdot_{(n)}$ subscript when redundant).

We use () to disambiguate where needed, and let application associate to the left and bind closer than abstraction – this allows ordinary conventions of rewriting and λ-calculus to be followed, *e.g.*, $\lambda xyz.xz(yz)$ denotes the rather unwieldy $\lambda^1([x](\lambda^1([y](\lambda^1([z]@^2(@^2(x,z), @^2(y,z)))))))$.

2.7 Example ($\lambda\beta$-calculus, readable CRS). Using the above abbreviations we can express the $\lambda\beta$-calculus in a more usual notation: the following CRS is exactly the same as the one in Example 2.5 above:

$$(\lambda x.Z(x))Y \to Z(Y) \qquad (\beta)$$

The substitution concept of CRS is reminiscent of a two-level λ-calculus in that metavariables are always 'metaapplied' to the list of terms that should be substituted into its body. Metavariables are therefore instantiated to 'substitute-abstractions' denoted $\underline{\lambda}\vec{x}.t$ and the resulting 'substitute-redexes' play the rôle of substitution.

2.8 Definition (substitution). A *valuation* σ is a function that maps each metavariable Z^n to a *substitute* $\underline{\lambda}(\vec{x}_{(n)}).t$ where the x_i are distinct and t is a preterm. Valuations are homomorphically extended to metaterms: $\sigma(t)$ denotes the result of first inserting $\sigma(Z)$ for each metavariable Z in t and then replacing all the resulting *substitutions* $(\underline{\lambda}(\vec{x}_{(n)}).t)(\vec{t}_{(n)})$ by $t[x_1 := t_1, \ldots, x_n := t_n]$ defined inductively by

$$\begin{aligned}
x_i[x_1 := t_1, \ldots, x_n := t_n] &\equiv t_i \\
y[x_1 := t_1, \ldots, x_n := t_n] &\equiv y \qquad y \notin \{\vec{x}\} \\
([y]t)[x_1 := t_1, \ldots, x_n := t_n] &\equiv [y](t[x_1 := t_1, \ldots, x_n := t_n]) \qquad y \notin \{\vec{x}\} \\
F^m(\vec{s})[x_1 := t_1, \ldots, x_n := t_n] &\equiv \\
&F^m(s_1[x_1 := t_1, \ldots, x_n := t_n], \ldots, s_m[x_1 := t_1, \ldots, x_n := t_n]) \\
Z^m(\vec{s})[x_1 := t_1, \ldots, x_n := t_n] &\equiv \\
&Z^m(s_1[x_1 := t_1, \ldots, x_n := t_n], \ldots, s_m[x_1 := t_1, \ldots, x_n := t_n])
\end{aligned}$$

(the last case exists because one metavariable may be mapped to another when we allow reduction of metaterms).

A lot of complexity is hidden in the requirement that valuations be 'homomorphically extended' to metaterms because of the risk of name clashes. This is solved by the following definition which turns out to be a sufficient restriction for avoiding name conflicts.

2.9 Definitions (safeness).

A. The *bound variables* of a preterm, $\mathrm{bv}(t)$, are defined inductively as $\mathrm{bv}(x) = \varnothing$, $\mathrm{bv}([x].t) = \{x\} \cup \mathrm{bv}(t)$, $\mathrm{bv}(F^n(\vec{t}_{(n)})) = \mathrm{bv}(Z^n(\vec{t}_{(n)})) = \mathrm{bv}(t_1) \cup \cdots \cup \mathrm{bv}(t_n)$.

B. The rewrite rule $p \to t$ is *safe for the valuation* σ if for p and t considered as preterms

$$\forall Z \in \mathrm{mv}(p)\ \forall x \in \mathrm{fv}(\sigma(Z)) : x \notin (\mathrm{bv}(p) \cup \mathrm{bv}(t))$$

C. The valuation σ is *safe with respect to itself* if

$$\forall Z, Z' : \mathrm{fv}(\sigma(Z)) \cap \mathrm{bv}(\sigma(Z')) = \varnothing$$

Thus a CRS inherits the implicit complexity of the λ-calculus renaming, and resolves it in the same way by a generalised form of *variable convention*:

2.10 Convention (variable convention). Any valuation used is assumed safe with respect to itself, and any rule it will be used with is assumed safe with respect to it.

Clearly a safe variant can be obtained for any valuation in any context by renaming of its bound variables. This renaming is harmless since we consider metaterms modulo renaming.

2.11 Definition (reduction). Given a rule $p \to t$. A *redex* of this rule is a term $\sigma(p)$ for some (safe) valuation σ, $\sigma(p) \to \sigma(t)$ is a *rewrite*, and for any context $C[\ ]$ without metavariables, $C[\sigma(p)] \to C[\sigma(t)]$ is a *reduction*.

If a CRS is considered with restricted terms then it is required that reduction of a restricted term yields a restricted term.

2.12 Remark (metaterm reduction). An interesting generalisation of the above is to consider reduction of metaterms. This is defined by replacing metaapplications in the metaterm with fresh function symbols and then reducing normally. In the first-order (TRS) case, metaconfluence is the same as what is known as 'confluence on open terms' which is subject to some interest for first-order explicit substitution calculi.[2]

3 Explicit Substitutes

In this section we show how the explicit substitution idea developed for named λ-calculus can be generalised to CRSs

With CRSs we can observe the same problem as with λ-calculus: the operational problem of using CRS reduction steps as a *complexity measure* is that the complexity of substitution depends on the size of the body of the substitute. The solution we propose is also the same as with λ-calculus: to perform substitution in a stepwise manner, or, put differently, to change metaredexes to use 'explicit substitutes', such that only *local term knowledge* is used in the rewriting (but not in the matching – we comment further on this at the end of the section).

First we demonstrate the idea as it was first developed for a particular CRS, namely the $\lambda\beta$-calculus. Then we identify a subclass of CRS that has 'explicit substitution' and show that every confluent CRS, in particular the functional ones, can be transformed into a (confluent) CRS in this class, thus working towards providing an equivalent with an operationally faithful complexity measure.

3.1 Example (λxgc-reduction as CRS).

As mentioned above, the idea of explicit substitution stems from studies of the λ-calculus. We demonstrate this by showing λxgc-reduction of (Bloo and Rose 1995) as part of a CRS with function symbols $\{\lambda^1, @^2, \Sigma^2\}$ and rewrite rules

$$(\lambda x.Z(x))Y \to \Sigma([x]Z(x), Y) \tag{b}$$

$$\Sigma([x]x, Y) \to Y \tag{xv}$$

$$\Sigma([x]Z, Y) \to Z \tag{xgc}$$

$$\Sigma([x]\lambda y.Z(x,y), Y) \to \lambda y.\Sigma([x]Z(x,y), Y) \tag{xab}$$

$$\Sigma([x]\,(Z_1(x))(Z_2(x)), Y) \to (\Sigma([x]Z_1(x), Y))(\Sigma([x]Z_2(x), Y)) \tag{xap}$$

Similar to the CRS $\lambda\beta$ we have to restrict the terms:

$$\lambda\text{xgc-terms:} \qquad t ::= x \mid \lambda([x]t) \mid @(t_1, t_2) \mid \Sigma([x]t_1, t_2)$$

Note again that reduction of restricted terms yields restricted terms.

It is instructive to compare this system to the definition of λxgc (Bloo and Rose 1995): they are identical except for syntax conventions (for instance the λxgc-term $x\langle x := \lambda y.y\rangle$ corresponds to $\Sigma([x]x, \lambda([y]y))$ in this paper), and due to the safeness condition on valuations there is no need to state the side condition $x \notin \mathrm{FV}(Z)$.

[2]The terminology is confusing because this has nothing to do with our notion of 'closed' since 'open' refers to the presence of *metavariables*, not free variables – but this is not a conflict in TRS where there are no metavariables.

The key observation in the development for the λ-calculus is that no knowledge of the 'depth' of terms is needed in order to reduce the rules. In the example rules above this is manifest in the fact that all metaapplications on both LHS and RHS have *only variable arguments*, in fact the *same variables on both sides!*

3.2 Definition (explicit substitution CRS). A CRS R is an *explicit substitution CRS* or simply *ESCRS*, if all metaaplications on the RHS of a rule occur in the form $Z(\vec{x})$ such that $Z(\vec{x})$ also occurs in the LHS of that same rule (this implies that the variables in $\vec{x}$ are distinct). Individual metaapplications on this form in RHSs will be called *explicit metaapplications*.

Another interesting observation which we can carry over from the study of explicit substitution for the λ-calculus is that substitution does not depend on the context in which it is created. This insight can be used to create an ESCRS automatically from a CRS in a manner that ensures that confluence is preserved by simply 'unfolding' the definition of substitutes (Definition 2.8) into the new rules.

3.3 Definition (CRS explicification). Given a CRS R with alphabet F^n_i. The ESCRS Rx is obtained by the steps listed in Figure 1. If R defined the relation $\rightarrow$ then we will denote the relation defined by Rx as $\xrightarrow[x]{}$; the subrelation consisting of only the introduction rules (with name (r-x)) is denoted $\xrightarrow[Ix]{}$, and the subrelation containing only the distribution and elimination rules (with names (x-∗)) is denoted $\xrightarrow[xE]{}$.

3.4 Example. The system λβx generated this way for the λβ-calculus with the rule

$$(\lambda x.Z(x))Y \rightarrow Z(Y) \tag{β}$$

is the following:[3]

$$\begin{aligned}
(\lambda x.Z(x))Y &\rightarrow \Sigma([x]Z(x), Y) && (\beta\text{-x})\\
\Sigma([x](Z_1(x))(Z_2(x)), Y) &\rightarrow (\Sigma([x]Z_1(x), Y))(\Sigma([x]Z_2(x), Y)) && (\text{x-@})\\
\Sigma([x]\lambda(Z(x)), Y) &\rightarrow \lambda(\Sigma([x]Z(x), Y)) && (\text{x-}\lambda)\\
\Sigma([x]x, Y) &\rightarrow Y && (\text{xv})\\
\Sigma([x]Z, Y) &\rightarrow Z && (\text{xgc})\\
\Sigma([x, y]Z(x, y), Y) &\rightarrow [y]\Sigma([x]Z(x, y), Y) && (\text{xma})
\end{aligned}$$

It is the same as the λxgc CRS shown above except that the abstraction distribution rule (xab) has been split into two steps: (x-λ) and (xma). As a consequence, a restricted term need not reduce to a restricted term (but another reductionstep can fix this). To prevent this, one can define the explicification more cautiously for CRSs with retricted terms (this means that some $\xrightarrow[xE]{}$ steps are forced to be followed by other $\xrightarrow[xE]{}$ steps). We will not do this in this paper since it is rather straightforward.

3.5 Proposition. For any CRS R, Rx is an ESCRS.

3.6 Remark. The procedure is idempotent: when applied to an ESCRS nothing is added since no non-explicit metaabstractions exist. Also notice that the resolution and distribution rules only depend on the alphabet.

[3]Generated automatically from (β) by a program developed by the second author.

Restricted terms. The restricted terms of the CRS Rx are defined by

$$t \;::=\; r \mid \Sigma([x]t_1, t_2)$$

where r ranges over terms of R and Σ is a new 2-ary symbol added to the alphabet.

Substitution introduction. For each rule (r) of R construct a new rule $(r\text{-}x)$ of Rx by replacing in the RHS all non-explicit metaapplications $Z^n(\vec{t})$ by the construction (containing n explicit metaapplications)

$$\Sigma([x_n]\Sigma([x_{n-1}]\cdots\Sigma([x_1]Z(\vec{x}_{(n)}), t_1), t_2)\cdots, t_n)$$

We shall abbreviate $\Sigma([x_n]\Sigma([x_{n-1}]\cdots\Sigma([x_1]Z(\vec{x}_{(n)}), t_1), t_2)\cdots, t_n)$ by $\Sigma([\vec{x}]Z(\vec{x}), \vec{t})$.

Stepwise substitution distribution. Add the rules

$$\Sigma([x][y]Y(x,y), Z) \to [y]\Sigma([x]Y(x,y), Z) \qquad \text{(xma)}$$

and for each possible F^m, add the rule

$$\Sigma([x]F^m(Z_1(x), \ldots, Z_m(x)), Y) \to F^m(\Sigma([x]Z_1(x), Y), \ldots, (\Sigma([x]Z_m(x), Y))) \qquad \text{(x-F)}$$

Substitution elimination. Add the rules

$$\Sigma([x]x, Z) \to Z \qquad \text{(xv)}$$

$$\Sigma([x]Y, Z) \to Y \qquad \text{(xgc)}$$

Figure 1: Construction of an ESCRS from a CRS.

The remainder of this section is devoted to showing that an ESCRS is a conservative extension of the original CRS.

3.7 Proposition. For any orthogonal R, $\xrightarrow[\mathrm{Ix}]{}$ and $\xrightarrow[\mathrm{xE}]{}$ are complete.

Proof sketch. Completeness of $\xrightarrow[\mathrm{Ix}]{}$ is easy by observing that the generated introduction rules cannot create any Ix-redexes and the orthogonality or R ensures that no redexes are destroyed, either. Completeness of $\xrightarrow[\mathrm{xE}]{}$ is shown by first establishing $\xrightarrow[\mathrm{xE}]{}$SN by defining a map dominating the longest reduction length; $\xrightarrow[\mathrm{xE}]{}$LC follows from a simple investigation of the critical pairs between the (xgc-n) rules and the other rules (this amounts to understanding that 'garbage collection' can be postponed without risk). $\square$

Next we relate single reductions and then build the components of the proof of multiple reduction equivalence (in this paper $\downarrow_{\mathrm{xE}}(t)$ denotes the unique $\xrightarrow[\mathrm{xE}]{}$-nf of t, and $\underset{\mathrm{xE}}{\twoheadrightarrow\!\!|}$ the restriction of $\underset{\mathrm{xE}}{\twoheadrightarrow}$ to reductions to normal forms).

3.8 Lemma (representation).

$$\downarrow_{\mathrm{xE}}(\Sigma([x_1]t, t_1)) \equiv \downarrow_{\mathrm{xE}}(t)[x_1 := \downarrow_{\mathrm{xE}}(t_1)]$$

Proof sketch. This is the 'substitution lemma' of the λ-calculus in disguise; we prove by induction on the number of symbols in the sequence $t, t_1, \ldots, t_n$ that $\downarrow_{\mathrm{xE}}(\Sigma([x_1, \ldots, x_n]t, t_1, \ldots, t_n)) \equiv \downarrow_{\mathrm{xE}}(t)[x_1 := \downarrow_{\mathrm{xE}}(t_1)] \cdots [x_n := \downarrow_{\mathrm{xE}}(t_n)]$.

$\square$

3.9 Lemmas (projection & injection). A.
$$\begin{tikzcd} s \arrow[r, "\mathrm{Ix}"'] \arrow[d, two heads, "\mathrm{xE}"'] & t \arrow[d, two heads, "\mathrm{xE}"'] \\ \downarrow_{\mathrm{xE}}(s) \arrow[r, dotted, two heads] & \downarrow_{\mathrm{xE}}(t) \end{tikzcd}$$
. B.
$$\begin{tikzcd} s \arrow[rr] \arrow[dr, dotted, "\mathrm{Ix}"'] & & t \\ & s' \arrow[ur, dotted, two heads, "\mathrm{xE}"'] & \end{tikzcd}$$
.

Proof sketch. Lemma 3.9A is shown by induction over the structure of Rx-metaterms, using Lemma 3.8; Lemma 3.9B is a simple consequence of the fact that the R-rules and Rx-introduction rules have the same redexes. $\square$

3.10 Theorem. *For* R*-metaterms* s, t *(i.e.,* s, t *are in* xE*-normal form when considered as* Rx*-terms):* $s \underset{\mathrm{x}}{\twoheadrightarrow} t$ *iff* $s \twoheadrightarrow t$.

Proof. **Case** $\Leftarrow$: Assume $s \twoheadrightarrow t$; the case then follows by induction on the length of the $\twoheadrightarrow$-reduction, using Lemma 3.9B in each step.

Case $\Rightarrow$: Assume $s \underset{\mathrm{x}}{\twoheadrightarrow} t$ and $s, t \in R$. We will do induction on the length of the $\underset{\mathrm{x}}{\twoheadrightarrow}$-reduction and prove

$$\begin{tikzcd} s \arrow[r, "\mathrm{x}"'] \arrow[d, equal] & s_1 \arrow[r, "\mathrm{x}"'] \arrow[d, two heads, "\mathrm{xE}"'] & \cdots\cdots \arrow[r, "\mathrm{x}"'] & s_{n-1} \arrow[r, "\mathrm{x}"'] \arrow[d, two heads, "\mathrm{xE}"'] & t \arrow[d, equal] \\ s \arrow[r, dotted, two heads] & \downarrow_{\mathrm{xE}}(s_1) \arrow[r, dotted, two heads] & \cdots\cdots \arrow[r, dotted, two heads] & \downarrow_{\mathrm{xE}}(s_{n-1}) \arrow[r, dotted, two heads] & t \end{tikzcd}$$

Each $\xrightarrow[\mathrm{x}]{}$-step is either $\xrightarrow[\mathrm{Ix}]{}$ or $\xrightarrow[\mathrm{xE}]{}$ for which we need Lemma 3.9A and confluence of $\xrightarrow[\mathrm{xE}]{}$, respectively. $\square$

An easy consequence of all this is the main result of this section.

3.11 Corollary. *If* R *is confluent then* Rx *is confluent.*

3.12 Discussion (fair complexity measure). We have now achieved what we set out to do: the reduction count is a fair complexity measure *provided we only count contraction complexity*: we have not considered the complexity of matching. This complexity involves two aspects that can be unbounded in the case of CRS:

- the time to locate a potential redex to reduce, and
- the time to search the redex to ensure that only the allowed free variables are present.

Both are inherent in CRS and require extensions to the formalism to solve[4].

4 Preservation of Strong Normalisation

In this section we investigate the termination behaviour of an ESCRS with respect to the corresponding CRS. We show that for a large subclass of CRSs, explicifying the reduction preserves strong normalisation of terms.

First we show by some examples that not all CRSs have the PSN property, then we define the subclass of *structure preserving CRSs* and show PSN for it.

4.1 Definition. We say that an explicification Rx has PSN if $\mathrm{SN}_{\mathrm{R}} \subseteq \mathrm{SN}_{\mathrm{Rx}}$.

4.2 Example. Consider the CRS R_1:

$$\begin{aligned} \mathsf{F}(a) &\to \mathsf{F}(a) \\ \mathsf{G}([x]X(x), Y) &\to X(\mathsf{F}(Y)) \end{aligned}$$

Then the term $\mathsf{G}([x]a, a)$ is SN since its only reduct is a. But in the ESCRS $\mathrm{R}_1\mathrm{x}$, the term $\mathsf{G}([x]a, a)$ can reduce to $\Sigma([x]a, \mathsf{F}(a))$ which reduces to itself.

We see that $\mathrm{R}_1\mathrm{x}$ does not have PSN. Two properties of R_1 can be indicated for causing this failure of PSN:

- R_1 has a circular rule
- R_1 has a rule that creates new structure as an argument of a meta-application.

The second property is absent in λ-calulus; indeed the proof of PSN for λxgc (see (Bloo and Rose 1995)) depends on the fact that newly created substitutions contain subterms that were already present in the term before these substitutions were created.

4.3 Example. Consider the CRS R_2, an extension of λ-calculus with the 3-ary symbol F and the rule

$$\mathsf{F}(\lambda x.X(x), \lambda y.Y(y), Z) \to X(Y(Z))$$

Then the term $\mathsf{F}(\lambda x.I, \lambda y.yy, \lambda z.zz)$ is strongly normalising since its only reduct is $\lambda x.I$. But in $\mathrm{R}_2\mathrm{x}$, $\mathsf{F}(\lambda x.I, \lambda y.yy, \lambda z.zz)$ reduces to $\Sigma([x]I, \Sigma([y]yy, \lambda z.zz))$ which reduces in three steps to $\Sigma([x]I, (\lambda z.zz)(\lambda z.zz))$; a term with Ω as subterm. Hence $\mathrm{R}_2\mathrm{x}$ does not have PSN. The reason for this is again that one of the rules of R_2 creates new structure as

[4]These extensions are studied in the second author's forthcoming thesis.

an argument of a metaapplication in the righthand side. This new structure is present in the righthand side of the corresponding substitution introduction rule but (due to a non-occurring variable) it may be absent in the righthand side of the non-explicit rule.

We saw that an ESCRS Rx may fail PSN if R has rules that create new structure as an argument of a metaapplication. We shall show that this is the only reason for failure of PSN. Therefore it seems fair to introduce a name for CRSs that do not create such new structure:

4.4 Definition. A CRS is called *structure preserving* if any argument of a metaapplication in the righthand side of a rule is a subterm of the lefthand side of that rule.

Note that λ-calculus is structure preserving whereas R_1 and R_2 are not.

We start our proof of PSN for structure preserving CRSs by considering (similar to PSN for λxgc in (Bloo and Rose 1995)) a calculus in between CRSs and ESCRSs.

4.5 Definition. Let R be a CRS and Rx the corresponding ESCRS.

- A substitution $\Sigma([x]t_1, t_2)$ contains *garbage* if $x \notin \mathrm{FV}(t_1)$.
- An Rx term is called *garbage-free* if it contains no garbage.
- $\rightarrow_{\mathrm{gf}}$ is a reduction on garbage-free terms of Rx defined by:

  $$t_1 \rightarrow_{\mathrm{gf}} t_2 \ \text{ iff } \ t_1 \xrightarrow[\mathrm{x}]{} \cdot \xrightarrow[\mathrm{xgc}]{}\!\!\!\!\rightarrow t_2$$

  that is, a $\rightarrow_{\mathrm{gf}}$ step consists of some $\xrightarrow[\mathrm{x}]{}$ reduction followed by complete garbage collection.
- $\downarrow_{\mathrm{xgc}}(t)$ is used to denote the xgc-normal form of t.

4.6 Lemma. Let R be a CRS.

A. $\Sigma([x]\Sigma([y]t_1, t_2), t_3) \overset{}{\twoheadleftarrow}\!\!\underset{\mathrm{xE}}{}\!\!\twoheadrightarrow \Sigma([y]\Sigma([x]t_1, t_3), \Sigma([x]t_2, t_3))$

B. If t is garbage-free and $t \xrightarrow[\mathrm{Ix}]{} s$ then $\downarrow_{\mathrm{xE}}(t) \xrightarrow{+} \downarrow_{\mathrm{xE}}(s)$.

C. $\mathrm{SN}_{\mathrm{R}} \subseteq \mathrm{SN}_{\mathrm{Rgf}}$

Proof. A. use Lemma 3.8 and the property of substitution that $t_1[x := t_2][y := t_3] \equiv t_1[y := t_3][x := t_2[y := t_3]]$.

B. Induction on the structure of t; t is garbage-free ensures that the redex cannot disappear in $t \xrightarrow[\mathrm{xE}]{} \downarrow_{\mathrm{xE}}(s)$.

C. We project garbage-free reductions to R-reductions: any garbage-free redcution is of the form $t_1(\xrightarrow[\mathrm{xE}]{} \cdot \xrightarrow[\mathrm{xgc}]{}\!\!\!\!\rightarrow)^* t_2 \xrightarrow[\mathrm{Ix}]{} \cdot \xrightarrow[\mathrm{xgc}]{}\!\!\!\!\rightarrow t_3(\xrightarrow[\mathrm{xE}]{} \cdot \xrightarrow[\mathrm{xgc}]{}\!\!\!\!\rightarrow)^* t_4 \xrightarrow[\mathrm{Ix}]{} \cdot \xrightarrow[\mathrm{xgc}]{}\!\!\!\!\rightarrow t_5 \cdots$ (since $\xrightarrow[\mathrm{xE}]{}$ is SN); then we project to get a reduction $\downarrow_{\mathrm{xE}}(t_1) \equiv \downarrow_{\mathrm{xE}}(t_2) \overset{+}{\twoheadrightarrow} \downarrow_{\mathrm{xE}}(t_3) \equiv \downarrow_{\mathrm{xE}}(t_4) \overset{+}{\twoheadrightarrow} \downarrow_{\mathrm{xE}}(t_5) \cdots$. Now if the first reduction is infinite then its projection is infinite too. □

Now we subdivide Rx-reduction into two disjoint parts, reductions that are related to garbage and the others:

4.7 Definition. For a CRS R, garbage reduction is defined as the contextual closure of the following rules:

A. if $s \xrightarrow[\text{xE}]{} s'$ and $x \notin FV(\downarrow_{xgc}(t))$ then $\Sigma([x]t, s) \xrightarrow[\text{xE}]{} \Sigma([x]t, s')$ is garbage-reduction.

B. if $x \notin FV(\downarrow_{xgc}(F^m(\vec{t}_{(m)})))$ then $\Sigma([x]F^m(\vec{t}_{(m)}), t) \rightarrow F^m(\Sigma([x]t_1, t), \ldots, \Sigma([x]t_m, t))$ is garbage-reduction.

C. if $x \notin FV(\downarrow_{xgc}([y]t))$ then $\Sigma([x][y]t, t') \rightarrow [y]\Sigma([x]t, t')$ is garbage-reduction.

D. $t \xrightarrow[\text{xgc}]{} t'$ is garbage-reduction.

Note that we use the free variable expression $FV(\downarrow_{xgc}(\cdot))$ to ensure that garbage collection does not create new garbage-reduction redexes.

Outside-garbage-reduction is defined as any reduction which is not garbage-reduction; this is equivalent to saying that the contracted redex has no descendant in the xgc-normalform.

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

4.8 Propositions. A. If $t \xrightarrow[\text{x}]{} s$ is garbage-reduction then $\downarrow_{xgc}(t) \equiv \downarrow_{xgc}(s)$.

B. If $t \xrightarrow[\text{x}]{} s$ is outside garbage then $\downarrow_{xgc}(t) \rightarrow \downarrow_{xgc}(s)$.

4.9 Definition. We say s *is body of a substitution in* t if for some t', x, $\Sigma([x]t', s)$ is a subterm of t. The predicate $\text{subSN}(t)$ should be read to be *all bodies of substitutions in* t *are strongly normalising for* $\xrightarrow[\text{x}]{}$*-reduction.*

The following lemma expresses our intuition about garbage-reduction.

4.10 Lemmas. A. If $\text{subSN}(t)$ and $t \xrightarrow[\text{x}]{} s$ is garbage-reduction, then $\text{subSN}(s)$.

B. If $\text{subSN}(t)$ then t is strongly normalising for garbage-reduction.

4.11 Definition. Define for all Rx-terms t, $\#\text{gf}(t)$ to be the maximum length of garbage-free reduction paths starting in $\downarrow_{xgc}(t)$. Note that $\#\text{gf}(t)$ can be infinite.

4.12 Theorem. *Let* R *be a structure preserving CRS, let* Rx *be the explicification of* R*, let* t *be an* Rx*-term.*

If $\#\text{gf}(t) < \infty$ *and* $\text{subSN}(t)$ *then* t *is strongly normalising for* $\xrightarrow[\text{x}]{}$*-reduction.*

Proof sketch. We use induction on $\#\text{gf}(t)$ using an extra induction over the term structure to show that the $\text{subSN}(\cdot)$ property is preserved for any reduct. This requires Lemma 4.10B, Proposition 4.8, and Lemma 4.10A. Furthermore it is essential that R is structure preserving since thereby a newly created body of a substitution was already present in the term just before the $\xrightarrow[\text{Ix}]{}$-step created it and has a smaller #gf-count; therefore the indcution hypothesis applies yielding that it is a strongly normalising term. □

4.13 Corollary (PSN for Rx). *An* R*-term is strongly normalising for* $\rightarrow$*-reduction if and only if it is strongly normalising for* $\xrightarrow[\text{x}]{}$*-reduction.*

Proof. **Case ⇒.** If t is an R-term then $\text{subSN}(t)$ and if t is strongly normalising for $\rightarrow$-reduction then by ??, $\#\text{gf}(t) < \infty$. Now use Theorem 4.12.

Case ⇐. By ?? infinite $\rightarrow$-reductions induce infinite $\xrightarrow[\text{x}]{}$-reductions. □

5 Conclusions

We have generalised the notion of explicit substitution to CRS, *cf.* Klop et al. (1993, sec.15(h)), and shown that the derived ESCRS systems inherit the essential properties of the original CRS, notably confluence and strong normalisation for a particular subset of CRSs.

Further directions. The study of ESCRS has proved to lead to many interesting considerations, mostly generalising problems known from explicit substitution for λ-calculus. Current work includes the following:

- In order to obtain a better complexity model (*cf.* remark Discussion 3.12) we need to restrict the definition of explicit metaapplication to require that all rule-bound variables are included as arguments! This means that the new rule (xgc-$\mathtt{n}$) becomes invalid; this cannot be solved within the CRS framework. We are investigating a solution where the CRS formalism is extended to allow special subpatterns that match only arbitrary bound variables and then this problem can be solved.
- Investigate if a variable-free (de Bruijn) notation is feasible.
- Derive abstract machines 'implementing' CRS through their ESCRS.
- Integration of cyclic substitution (Rose 1992, Ariola and Klop 1994).
- Investigate whether the technique is applicable to other forms of higher order rewriting, in particular the relation to the notion of 'substitution calculus'.
- Finally, the above serve to highlight the relation to functional programming language, in particular supercombinator reduction.

References

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

Aczel, P. (1978). A general Church-Rosser theorem. *Technical report*. Univ. of Manchester.

Ariola, Z. M. and Klop, J. W. (1994). Cyclic lambda graph rewriting. *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press. Paris, France. pp. 416–425.

Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics*. Revised edn. North-Holland.

Bloo, R. and Rose, K. H. (1995). Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. *CSN '95 – Computer Science in the Netherlands*. ⟨URL: *ftp://ftp.diku.dk/diku/semantics/papers/D-246.ps*⟩

de Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Koninklijke Nederlandse Akademie van Wetenschappen, Series A, Mathematical Sciences* 75: 381–392. Also chapter C.2 of (Nederpelt, Geuvers and de Vrijer 1994).

Henderson, P. (1980). *Functional Programming—Application and Implementation*. Prentice-Hall.

Klop, J. W. (1980). *Combinatory Reduction Systems*. PhD thesis. University of Utrecht. Also available as Mathematical Centre Tracts 127.

Klop, J. W., van Oostrom, V. and van Raamsdonk, F. (1993). Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science* 121: 279–308.

Landin, P. J. (1964). The mechanical evaluation of expressions. *Computer Journal* 6: 308–320.

Nederpelt, R. P., Geuvers, J. H. and de Vrijer, R. C. (eds) (1994). *Selected Papers on Automath*. Vol. 133 of *Studies in Logic*. North-Holland.

Nipkow, T. (1991). Higher-order critical pairs. *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press. Amsterdam, The Netherlands. pp. 342–349.

Rose, K. H. (1992). Explicit cyclic substitutions. *In* Rusinowitch, M. and Rémy, J.-L. (eds), *CTRS '92—3rd International Workshop on Conditional Term Rewriting Systems*. Number 656 in *LNCS*. Springer-Verlag. Pont-a-Mousson, France. pp. 36–50. ⟨URL: *ftp://ftp.diku.dk/diku/semantics/papers/D-143.ps*⟩

Rose, K. H. and Bloo, R. (1995). A named lambda calculus with explicit substitution and garbage collection. *Semantics note*. DIKU (University of Copenhagen). In preparation. ⟨URL: *ftp://ftp.diku.dk/diku/semantics/papers/D-?.ps*⟩

van Oostrom, V. and van Raamsdonk, F. (1995). Weak orthogonality implies confluence: the higher-order case. *Technical Report CS-R9501*. CWI.