Combinatory Reduction Systems with Explicit Substitution

Kristoffer H. Rose
*kris@diku.dk**

Extended Abstract, September 1, 1995

Abstract

We show how the idea of explicit substitution can be applied to give a measure of the computational complexity of rewriting in combinatory reduction systems (CRS) by constructing an extension Rx with explicit substitution for any CRS R. We prove that Rx is confluent if R is orthogonal.

Key words: orthogonal combinatory reduction systems, explicit substitution, confluence.

1 Introduction

The λ-calculus is a useful basis for denotational models of the computable functions. However, the tradition of counting β-reduction steps $(\lambda x.M)N \to M[x := N]$, where $M[x := N]$ is metanotation for the λ-term obtained by substituting in M all (unbound) occurrences of x by N, does not account for 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 real 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 (de Bruijn 1978, Abadi, Cardelli, Curien and Lévy 1991, Kamareddine and Nederpelt 1993, Lescanne 1994) 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.

*DIKU, University of Copenhagen, Universitetsparken 1, DK–2100 København Ø; ⟨URL: http://www.diku.dk/~kris/⟩.

Overview of the paper. 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) as well as some knowledge of higher order rewriting (we will use the CRS formalism as described by Klop, van Oostrom and van Raamsdonk (1993)).

2 Combinatory Reduction Systems

The intention with CRS is to add 'bound variables' and the associated concept of 'substitution' to TRS, thus combining TRS with λ-calculus – in fact, the initial treatment of Klop (1980) (as well as the investigation of Aczel (1978)) was concerned with 'definiable extensions of the λ-calculus'.

This section gives an introduction to the notion of *Combinatory Reduction System* (CRS). We first provide some intuition and then proceed with the needed formal results. The presentation follows the (recommended) (Klop et al. 1993) closely, with a slight change of notation to facilitate induction over syntax.

With start with the complete definition followed by a discussion of some important issues, then we state the results we will be using.

2.1 Definition (Combinatory Reduction System). A *CRS* is defined by

a. A set of *function symbols*, *aka* the *alphabet*, denoted $F, G, \ldots$ (and sometimes greek letters and weird symbols); each with a *fixed arity* added as superscript (*i.e.*, F^n).

b. The corresponding *metaterms*, denoted $t, s, \ldots$, have the form

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

called *variables*, *metaabstractions*, *constructions*, and *metaapplications*, respectively, where $x, y, z, \ldots$ are used to denote (simple) *variables* and Z, Y, X (only!) to denote *metavariables*, each with an explicit arity superscript as indicated (and possibly marks, subscripts, *etc*). We extract the *set of metavariables* of a term with $\mathrm{mv}(\cdot)$.

Furthermore, metaterms must always be *closed*: have no free variables, *i.e.*, $\mathrm{fv}(t) = \varnothing$ where $\mathrm{fv}(x) = \{x\}$, $\mathrm{fv}([x].t) = \mathrm{fv}(t) \backslash \{x\}$, $\mathrm{fv}(F^n(t_1, \ldots, t_n)) = \mathrm{fv}(t_1) \cup \cdots \cup \mathrm{fv}(t_n)$, and $\mathrm{fv}(Z^n(t_1, \ldots, t_n)) = \mathrm{fv}(t_1) \cup \cdots \cup \mathrm{fv}(t_n)$.

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

d. *Rewrite rules*, written $p \longrightarrow t$, with the following restrictions:

- The LHS (left hand side) p must be a *linear pattern*: it should be a construction, all metavariables should be distinct, and each metaabstraction should have paiwise distinct variables as parameters.
- All metaapplications on the RHS (right hand side) t must use metavariables that occurred in the LHS with the same arity.

2.2 Remark (Term Rewriting Systems). Any term rewrite system (TRS) is also a CRS containing no metaabstractions and only metavariables of arity zero (namely the TRS variables). The restriction to terms of the defined relation is the same as defined by the TRS, so in general we call any metaterm containing no metaabstractions for a *term*.

2.3 Example (λβ-calculus). The ordinary untyped λβ-calculus is described by the CRS with function symbols $\{\lambda^1, @^2\}$ and the single rule $@^2(\lambda^1([x]Z^1(x)), Y^0()) \to Z^1(Y^0())$ This 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 $(Z^1(x))[x := Y^0()]$.

2.4 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-arity symbols.
- $[x, y, z]t$ abbreviates $[x][y][z]t$.
- $Fxyz.t$ similarly abbreviates 'F-abstraction' $F([x][y][z]t)$.
- st abbreviates 'application' $@(s, t)$ (when it has no other interpretation) where $@^2$ is then included as a function symbol.
- $\vec{x}_n$, $\vec{Z}_n$, 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 use $()$ to disambiguate where needed, and application binds closer than abstraction. This allows ordinary conventions to be followed: $\lambda xyz.xz(yz)$ denotes $\lambda^1([x][y][z]@^2(@^2(x, z), @^2(y, z)))$.

2.5 Example (readable λ-calculus). Using the above abbreviations we can express the λ-calculus in a more usual notation: the following CRS is exactly the same as the previous:

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

(but we could not have written the RHS as ZY, of course).

2.6 Example (combinatory logic). The class of *applicative* TRS is naturally expressed using the above abbreviation. An example is combinatory logic:

$$SXYZ \to (XZ)(YZ) \qquad (S)$$
$$KXY \to X \qquad (K)$$

(but we should not write the RHS of (S) as $XZ(YZ)$ because that can be interpreted as $@(X, Z(@(Y, Z)))$ instead of the intended $@(@(X, Z), @(Y, Z))$.)

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

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

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

2.8 Remark. 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 requiring that each combination or a rule and valuation is *safe* by renaming.[1]

2.9 Definition (reduction). Given a rule $s \to t$. A *redex* of this rule is a term $\sigma(s)$ for some valuation σ, $\sigma(s) \to \sigma(t)$ is a *rewrite*, and for any context $C[\,]$, $C[\sigma(s)] \to C[\sigma(t)]$ is a *reduction*.

Our attention in this paper is restricted to the orthogonal CRSs because confluence is then secured.

2.10 Definitions (orthogonal CRS).
a. A CRS is *nonoverlapping* if for any redex r containing another, this other always appears as a subterm of something matched to a variable.
b. A CRS is *left-linear* if all LHSs contain at most one occurrence of each metavariable.
c. A CRS is *orthogonal* if it is left-linear and nonoverlapping.

2.11 Theorem (confluence of orthogonal CRS). *Any orthogonal CRS is confluent.*

3 Explicit Substitutes

In this section we show how the substitutes of CRSs can be made explicit.

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 have 'explicit substitution' and show that every (orthogonal) CRS can be transformed into a (confluent) CRS in this class, thus providing an operationally equivalent system for which the complexity of reduction can be investigated in a realistic way.

3.1 Example ($\lambda\beta$-calculus). As mentioned above, the idea of explicit substitution stems from studies of the λ-calculus. We demonstrate this by showing λxgc of Rose and Bloo (1995): it uses variables and can thus be written as the CRS (with explicit application operator $@^2$)

$$\begin{aligned}
@((\lambda x.Z(x)),Y) &\to \Sigma([x]Z(x),Y) && \text{(b)} \\
\Sigma([x]x,Y) &\to Y && \text{(xv)} \\
\Sigma([x]Z,Y) &\to Z && \text{(xgc)} \\
\Sigma([x]\lambda y.Z(x,y),Y) &\to \lambda[y](\Sigma([x]Z(x,y),Y)) && \text{(xab)} \\
\Sigma([x]\,@(Z_1(x),Z_2(x)),Y) &\to @(\Sigma([x]Z_1(x),Y),\Sigma([x]Z_2(x),Y)) && \text{(xap)}
\end{aligned}$$

[1] The main complexity in the author's CRS interpreter is this renaming.

It is instructive to compare this system to the definition of λxgc (Rose and Bloo 1995): they are identical except for syntax conventions and an important point: the requirement that all terms are closed prohibits free variables in CRS rules which means that all free variables must be 'hidden' inside metavariables (to which they are not given as parameters). This means that we cannot distinguish between substituting a free variable (λxgc rule (xvgc)) and the more general garbage collection rule (xgc).

Notice that this is not an orthogonal CRS: it has an overlap between the first and last rule with terms of the form $\Sigma([x]\underline{@(\lambda[y]Z_1(x,y),Z_2(x))},Y)$. However, it is confluent as shown in Rose and Bloo (1995) (and below as a consequence of the general case).

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 metaabstractions on both LHS and RHS have *only variable arguments*, in fact the *same variables on both sides!*

Analogously to the 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 is also the same: to perform substitution in a stepwise manner, or, put differently, to change metaredexes to use *explicit substitutes*, such that only 'superficial term knowledge' is used in the rules, and it is obtained in the same manner, by 'unfolding' the definition of substitutes into the rule set.

3.2 Definition (explicit substitution CRS). A CRS R is an *explicit substitution CRS* or simply *XCRS,* if all metaapplications on the RHS match the form $[x_1,\ldots,x_n]Z(x_1,\ldots,x_n)$. Individual metaabstractions on this form in RHSs will be called *explicit metaabstractions*.

3.3 Remark. Notice that even if it does not quite look so, the above system is on this form. This comes from the fact of CRS that *only bound variables are mentioned*. Specifically, the underlined part of $\lambda[y](\Sigma(\underline{[x]Z(x,y)},Y))$, the RHS of (xab), is of the form $[x]Z(x)$, because the y variable is external to this part. And renaming does not matter, so it is also of the form $[y]Z(y)$, *etc.*

Clearly the λxgc-calculus presented above is an XCRS. The interesting observation which we can carry over from the study of that calculus is that substitution does not depend on the context in which is is created.

This insight can be used to create an XCRS automatically from a CRS in a manner that ensures that the properties are preserved. The rest of this section details that construction.

3.4 Definition (CRS explicification). Given a CRS R with alphabet F_i^n. The CRS 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 $\xrightarrow[xE]{}$.

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

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

Substitution introduction. For each rule (r) of the CRS construct a new rule (r-x) by replacing in the RHS all non-explicit metaabstractions $Z^n(\vec{t}_n)$ by the (explicit) metaabstraction

$$\Sigma^{n+1}([\vec{x}_n]Z(\vec{x}_n), \vec{t}_n)$$

where Σ^{n+1} is a new function symbol added to the alphabet (thus there will be one new symbol for each metavariable arity).

Stepwise substitution distribution. For each possible pair of Σ^{n+1} and F^m, add the rule

$$\begin{aligned}&\Sigma^{n+1}([\vec{x}_n]F^m(Z_1(\vec{x}_n), \ldots, Z_m(\vec{x}_n)), \vec{Y}_n)\\ &\qquad \to F^m(\Sigma^{n+1}([\vec{x}_n]Z_1(\vec{x}_n), \vec{Y}_n), \ldots, (\Sigma^{n+1}([\vec{x}_n]Z_m(\vec{x}_n), \vec{Y}_n)))\end{aligned} \qquad (x\text{-}F\text{-}n)$$

(n rules are added for each function symbol F^m).

Substitution elimination. For each of these new Σ^{n+1} add the rules

$$\begin{aligned}\Sigma^{n+1}([\vec{x}_n]x_1, \vec{Z}_n) &\to Z_1 && (xv\text{-}n\text{-}1)\\ &\vdots && \vdots\\ \Sigma^{n+1}([\vec{x}_n]x_n, \vec{Z}_n) &\to Z_n && (xv\text{-}n)\\ \Sigma^{n+1}([\vec{x}_n]Y, \vec{Z}_n) &\to Y && (xgc\text{-}n)\\ \Sigma^{n+1}([\vec{x}_n]Y(\vec{x}_n, y), \vec{Z}_n) &\to [y]\Sigma^{n+1}([\vec{x}_n]Y(\vec{x}_n, y), \vec{Z}_n) && (xma\text{-}n)\end{aligned}$$

(a total of $n + 2$ rules for each Σ^{n+1} are added).

Figure 1: Explicification steps.

is the following:[2]

$$\begin{aligned}(\lambda x.Z(x))Y &\to \Sigma([x]Z(x), Y) && (\beta\text{-}x)\\ \Sigma([x](Z_1(x))(Z_2(x)), Y) &\to (\Sigma([x]Z_1(x), Y))(\Sigma([x]Z_2(x), Y)) && (x\text{-}@\text{-}1)\\ \Sigma([x]\lambda(Z(x)), Y) &\to \lambda(\Sigma([x]Z(x), Y)) && (x\text{-}\lambda\text{-}1)\\ \Sigma([x]x, Y) &\to Y && (xv\text{-}1\text{-}1)\\ \Sigma([x]Z, Y) &\to Z && (xgc\text{-}1)\\ \Sigma([x, y]Z(x, y), Y) &\to [y]\Sigma([x]Z(x, y), Y) && (xma\text{-}1)\end{aligned}$$

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-1) and (xma-1).

3.6 Remark. Clearly Rx is always an XCRS: the procedure is idempotent when applied to an XCRS because nothing is added when no non-explicit metaabstractions exist. Also notice that the resolution and distribution rules only depend on the alphabet.

The following general result is easy to show:

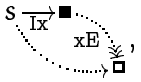
[2] In fact generated automatically from (β) by a program developed by the author.

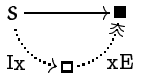
3.7 Proposition. For any orthogonal R, the derived $\overrightarrow{\overrightarrow{Ix}}$ and $\overrightarrow{\overrightarrow{xE}}$ are complete.

Proof sketch. Completeness of $\overrightarrow{\overrightarrow{Ix}}$ 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 $\overrightarrow{\overrightarrow{xE}}$ is shown by first establishing $\overrightarrow{\overrightarrow{xE}}$SN by defining a map dominating the longest reduction length; $\overrightarrow{\overrightarrow{xE}}$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). □

We will let $\downarrow_{xE}(t)$ denote the unique $\overrightarrow{\overrightarrow{xE}}$-nf of the term t, and $\overrightarrow{\overrightarrow{xE}}^{\twoheadrightarrow}$ the restriction of $\overrightarrow{\overrightarrow{xE}}^{\twoheadrightarrow}$ to reductions to normal forms.

Next we relate single reductions and then build the components of the proof of multiple reduction equivalence.

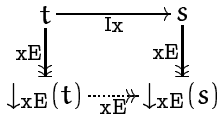
3.8 Proposition. For s, a. 

, and b. 

.

3.9 Lemma (representation). $\downarrow_{xE}(\Sigma^{n+1}([\vec{x}_n]t, \vec{t}_n) \equiv \downarrow_{xE}(t)[x_1 := \downarrow_{xE}(t_1), \ldots, x_n := \downarrow_{xE}(t_n)]$.

Proof sketch. This proof is the 'substitution lemma' of the λ-calculus in disguise and is shown in the same manner. □

3.10 Lemma (projection). For all t, 

.

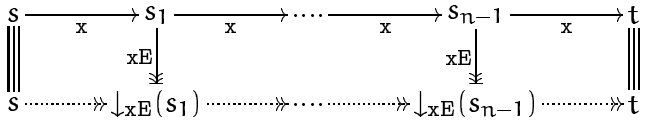
Proof sketch. Induction over the structure of Rx-metaterms. □

With this we can show the main theorem of the paper:

3.11 Theorem. *For* R*-metaterms* a, t*:* $s \overrightarrow{\overrightarrow{x}}^{\twoheadrightarrow} t$ *iff* $s \twoheadrightarrow t$.

Proof. **Case** ⇐: Assume $s \to t$; the case then follows by induction on the length of the $\overrightarrow{\overrightarrow{x}}^{\twoheadrightarrow}$-reduction, using 3.8b in each step.

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

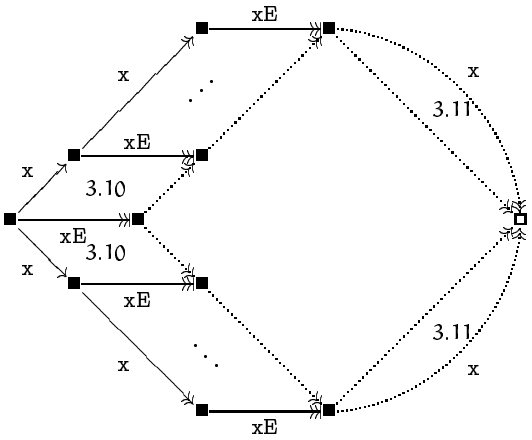


Each $\overrightarrow{\overrightarrow{x}}^{\twoheadrightarrow}$-step is either $\overrightarrow{\overrightarrow{Ix}}^{\twoheadrightarrow}$ or $\overrightarrow{\overrightarrow{xE}}^{\twoheadrightarrow}$ for which we need 3.10 and confluence of $\overrightarrow{\overrightarrow{xE}}$, respectively. □

An easy consequence of all this is that 'modular' properties of R are preserved for Rx, in particular this is true for confluence, the main result of the paper:

3.12 Corollary. *If* R *is orthogonal then* Rx *is confluent.*

Proof. The following diagram exploits $\twoheadrightarrow_{\Diamond}$ to show $\overline{\overline{\twoheadrightarrow}}^{x}_{\Diamond}$:



(The technique is called the 'interpretation method' by Hardin and Lévy (1990).) □

4 Conclusions

We have generalised the notion of explicit substitution to CRS as requested by Klop et al. (1993, sec.15(h)).

Further directions. Current work includes generalising other results obtained for λxgc to XCRS:

- Preservation of SN: show that $\mathrm{SN}_{\mathrm{Rx}} = \mathrm{SN}_{\mathrm{R}}$.
- Derive abstract machines for orthogonal CRS through their XCRS.
- Investigate if a variable-free (de Bruijn) notation is feasible.
- Investigate whether cyclic substitution (Rose 1992, Ariola and Klop 1994) can be integrated.
- Investigate whether the technique is applicable to other forms of higher order rewriting, in particular the relation to the 'substitution calculus' of van Oostrom and van Raamsdonk (1995).
- Also the relation to funtional programming language, in particular supercombinator reduction, is under investigation.

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.* Paris, France. pp. 416–425.

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

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.

de Bruijn, N. G. (1978). A namefree lambda calculus with facilities for internal definition of expressions and segments. *TH-Report 78-WSK-03*. Dept. of Mathematics, Technological University Eindhoven. Netherlands.

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.

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

Kamareddine, F. and Nederpelt, R. (1993). On stepwise explicit substitution. *International Journal of Foundations of Computer Science* 4(3): 197–240.

Klop, J. W. (1980). *Combinatory Reduction Systems*. Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam.

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

Lescanne, P. (1994). From $\lambda\sigma$ to $\lambda\upsilon$: a journey through calculi of explicit substitutions. *POPL '94—21st Annual ACM Symposium on Principles of Programming Languages*. Portland, Oregon. pp. 60–69.

Rose, K. H. (1992). Explicit cyclic substitutions. *In* M. Rusinowitch and J.-L. Rémy (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). Named lambda calculi with explicit substitution and garbage collection. *Semantics note*. DIKU (University of Copenhagen). In preparation; available through ⟨URL: http://www.diku.dk/~kris/research.html⟩.

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