Lambda Calculus with Explicit Recursion

Zena M. Ariola & Jan Willem Klop

CIS-TR-96-04
April 1996

Department of Computer and Information Science
University of Oregon

Lambda Calculus with Explicit Recursion

Zena M. Ariola
Computer and Information Science Department
University of Oregon. Eugene, OR 97401, USA
email: ariola@cs.uoregon.edu

Jan Willem Klop
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
Department of Mathematics and Computer Science
Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam
email: jwk@cwi.nl

Abstract

This paper is concerned with the study of λ-calculus with explicit recursion, namely of cyclic λ-graphs. The starting point is to treat a λ-graph as a system of recursion equations involving λ-terms, and to manipulate such systems in an unrestricted manner, using equational logic, just as is possible for first-order term rewriting. Surprisingly, now the confluence property breaks down in an essential way.

Confluence can be restored by introducing a restraining mechanism on the 'substitution' operation. This leads to a family of λ-graph calculi, which can be seen as an extension of the family of $\lambda\sigma$-calculi (λ-calculi with explicit substitution). While the $\lambda\sigma$-calculi treat the let-construct as a first-class citizen, our calculi support the letrec, a feature that is essential to reason about time and space behavior of functional languages.

Keywords & Phrases: lambda calculus, recursion, infinitary lambda calculus, term graph rewriting.
Note: The research of the first author is supported by NSF grant CCR-94-10237. The research of the second author is partially supported by ESPRIT BRA-6454 Confer. Further support was provided by ESPRIT Working Group 6345 Semagraph. A shorter version of this paper appears in the Proceedings of LICS 94 as "Cyclic Lambda Graph Rewriting".

Introduction

It is important to base the activities of programming, of writing a compiler, and of implementing the run-time support for a programming language on mathematical concepts. This can be done, without introducing too much mathematical machinery, with a *rewriting* or *calculator* approach that consists of mechanically applying a set of rewrite or simplification rules to a program. This method provides a *programmer*, a *compiler writer*, and an *implementor* with a sound basis to present, check, and try out their ideas. However, the usefulness of this abstract framework relies on how faithfully it models reality. In that respect, note that while *cyclic structures* are ubiquitous in a program development system [PJ87], current models of computation, such as the λ-calculus [Bar84] and term rewriting systems (Dershowitz et al.

[DJ90]; Klop [Klo92]), do not allow reasoning about them. As such, these models do not constitute the right computational vehicle for reasoning about the *time* and *space* behavior of a program.

Cycles occur in the representation of data structures. Consider the following data structure definition written in the lenient language Id [Nik91]:

$$\{\texttt{ones} = 1 : \texttt{ones} \\ \texttt{in ones}\} \ .$$

(A note on syntax: the construct $\{\cdots \texttt{in} \cdots\}$ represents a block expression, which consists of a group of unordered bindings and an expression which is written following the keyword in; : is the Id list constructor.) This is usually expressed in the λ-calculus using the fixed point combinator Y, whose behavior is captured by the following rewrite rule:

$$\texttt{Y}M \longrightarrow M(\texttt{Y}M) \ .$$

Thus, the above data structure ones becomes:

$$\texttt{Y}(\lambda x.1 : x) \ ,$$

which leads to the following rewriting ($\longrightarrow$ reads as "rewrites or reduces to"):

$$\texttt{Y}(\lambda x.1 : x) \longrightarrow (\lambda x.1 : x)(\texttt{Y}(\lambda x.1 : x)) \longrightarrow 1 : (\texttt{Y}(\lambda x.1 : x)) \ .$$

The above sequence of rewritings suggests that ones is represented in terms of a cons cell, with the head containing 1 and the tail pointing to the computation that delivers the rest of the list. However, this is not what happens in practice, ones is represented in terms of a single cons cell, with the tail pointing to the cons cell itself. Thus, access to any element of the list will only involve unwinding the data structure and no further computation. As introduced by Turner [Tur79], this representation can be captured in the following way: instead of the above Y-rule, use its optimized version, which involves a cycle (see Figure 1, in which @ stands for application).

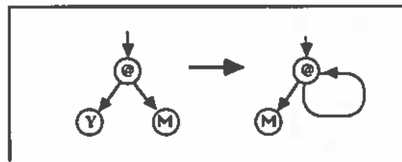


Figure 1: Cyclic Y-rule.

Cyclic structures do not only occur in non-strict languages. In a strict language, one can create them with side-effect operations. For example, in Standard ML [Har86] the data

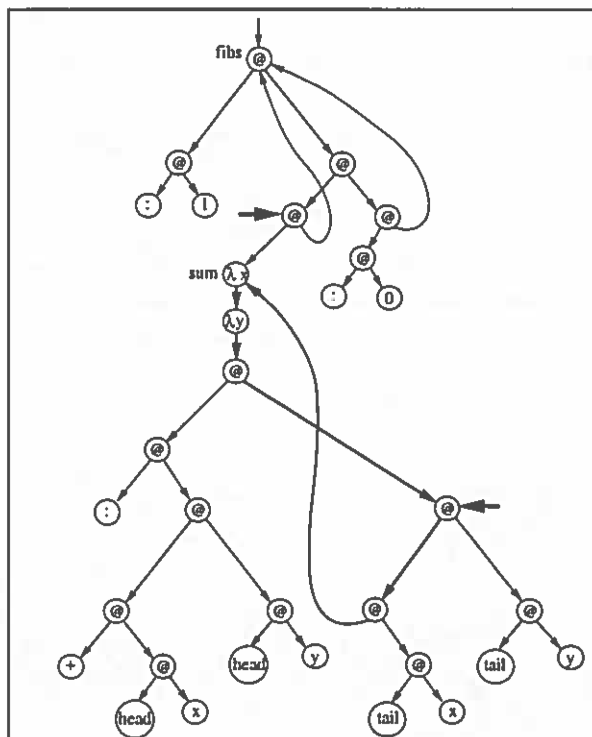


Figure 2: Cyclic lambda graph for computing the sequence of Fibonacci numbers.

structure ones can be expressed as follows:

```
datatype reflist = CONS of int * reflist ref | NIL;

(* Values of reflist have the form Cons(i, j), for i, an integer value,
    and j, a reference to a reflist value, or NIL. *)

let val x = ref(NIL);    (* associates x with a reference to a location containing NIL *)
  in
    x := CONS(1,x);       (* change the value x refers to *)
    x;                    (* return the reference *)
end .
```

Cycles also occur in the data structure representing the run-time environment when implementing recursive functions in either strict or non-strict languages. For example, the local environment created by the following Scheme expression

```
(letrec
    ((fact (lambda(n)
              (if (zero? n) 1
                        (* n (fact (-n 1)))))))
...)
```

contains a circularity, which is usually implemented using assignments, as described in the Scheme report [CR90][1]. Thus, dealing with cycles is desirable if one wants to discuss issues of data representation, and it becomes necessary if one wants to provide a computational model that supports reasoning about both functions and state. Moreover, capturing cycles is not only important for reasoning about run-time issues, but it is also important for reasoning about *compilation* and *optimization* of programs, as is discussed next.

Consider the sequence of Fibonacci numbers written in a lazy language (*e.g.,* Haskell [HPJW+92]) as follows:

$$\begin{array}{l} \texttt{let fibs} = \texttt{1} : \underbrace{\texttt{sum fibs (0:fibs)}}_{\rho_1} \\ \quad \texttt{sum} = \texttt{\\x y->(head x} + \texttt{head y)} : \underbrace{\texttt{sum (tail x) (tail y)}}_{\rho_2} \\ \texttt{in fibs} \end{array}$$

(The form "`\ x y -> e`" is Haskell's syntax for a lambda abstraction. As before, : is the list constructor; `sum fibs (0:fibs)` performs the addition of the `fibs` sequence and the sequence `0:fibs`.) The corresponding cyclic graph is displayed in Figure 2. In order to share the work among all invocations of a function and all accesses to a data structure, it makes sense to perform computations that occur inside a function body or inside a data structure at compile time. Specifically, we would like to reduce the redexes (*i.e.,* reducible expression) ρ_1 and ρ_2 in the Fibonacci program above. These redexes are indicated with an arrow in Figure 2. Both redexes express the application of a function to the arguments; their reduction corresponds to what in the literature has been referred to as *inlining*, *β-contraction* or *unfolding* [App92]. However, they are not usual redexes, since they are in a cycle. As such, their reduction is not at all obvious. In fact, as shown in this paper a naive approach will lead to a non-confluence result, *i.e.,* depending on how we apply the above transformations we get different programs. The lack of confluence has both theoretical and practical impacts. From a theoretical point of view, proofs that the above transformations are correct might become harder. From a practical point of view, non-confluence means that the order of application could ultimately have an impact on efficiency. Thus, a rigorous study of the reasons that cause confluence to fail is beneficial for getting a better grasp on how to apply program transformations, including Wadler's deforestation technique [Wad90], partial evaluation [JGS93], and the Burstall and Darlington unfold/fold [BD77]. These last transformations introduce new cycles by identifying previously encountered expressions. The difficulties of reasoning about circular programs is reflected by the fact that, in general, these transformations do not preserve total correctness.

In conclusion, while cyclic structures have been extensively used by implementors and compiler writers, their theoretical study is new, and up to now, has been limited mostly to first-order theories [SPvE93]. Typical results are confluence and correctness with respect to either infinitary term rewriting [KKSdV95b] or with respect to finite approximations [Ari93, AA95]. An operational treatment of cycles in the context of lazy (*i.e.,* call-by-need) languages

[1] Rosaz in [Ros92a] argued that the same efficiency can be gained by implementing recursion using suitable versions of the Y combinator but at the expenses of more complex analysis.

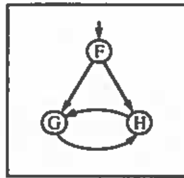


Figure 3: Horizontal sharing.

is given by Purushothaman et al. [PS92] and Launchbury [Lau93]. Purushothaman et al. provide a restricted form of cyclic structures, namely, those that express 'vertical sharing'. (Informally, a graph has only vertical sharing if there are no two different acyclic paths starting from the root to the same node.) For example, Purushothaman et al. does not deal with the cyclic graph of Figure 3, which expresses 'horizontal sharing.' Thus, their treatment of cycles is inadequate to describe the heap structure of a program execution. Launchbury deals with both kinds of sharing. However, he does not provide an equational theory, as such his work is not useful for expressing and reasoning about compiler transformations, or for studying the effect of different run-time implementation strategies.

The paper is organized as follows. We start, in Section 1, by introducing our approach to cycles that is based on systems of recursion equations. Until Section 8, we restrict our attention to systems of recursion equations involving λ-calculus extended with constants. No nesting of equations is admitted. In Section 2, we informally show how to manipulate such systems in an unrestricted manner, using equational logic, just as is possible for first-order term rewriting [AK95]. This naive way of rewriting, called the $\lambda\Theta$-calculus, is formally introduced in Section 3. Surprisingly, as shown in Section 4, the confluence property of $\lambda\Theta$ breaks down in an essential way. We point out, in Section 5, that the same phenomenon occurs in the infinitary lambda calculus developed by Kennaway et al. [KKSdV95a]. We discuss, in Section 6, another source of non-confluence that does not arise in the infinitary lambda calculus. In Section 7, we show soundness of $\lambda\Theta$ with respect to the infinitary lambda calculus, and how to restore confluence by controlling or restricting the operations on the recursion equations. We also point out that the $\lambda\mu$-calculus (*i.e.*, the λ-calculus extended with the μ-rule) which embodies much of cyclic λ-graph rewriting is confluent. In Section 8, we extend our framework to include nesting of recursion equations. We discuss a family of calculi, called $\lambda\phi$, that incorporate the λ-calculus, the $\lambda\mu$-calculus, ordinary first-order term rewriting, term graph rewriting, vertical and horizontal sharing. The $\lambda\phi$-calculi that we discuss can be seen as an extension of the family of $\lambda\sigma$-calculi (λ-calculi with explicit substitution) [HL89, ACCL91, Cur93, Les94]. While these concern acyclic expressions, our systems support cyclic substitutions. A related system for 'explicit cyclic substitutions' has been given by Rose [Ros92b]. The advantage, however, of the λ-graph calculi proposed in this paper is their proximity to the λ-calculus. We conclude the paper with future directions of research.

1. Systems of recursion equations over the λ-calculus

In the first part of the paper (Sections 1-7) we will consider systems of recursion equations over the λ-calculus. Thus we may write:

$$\alpha = \lambda x.x\alpha \ .$$

This is an object whose unwinding is an 'infinite normal form', also known as a Böhm-tree [Bar84]. We also may consider mutual recursion as in

$$\alpha = (\lambda x.\delta xx)\alpha, \ \delta = (\lambda y.\alpha y)\delta \ .$$

We will always use $\alpha, \delta, \cdots$, for recursion variables. For the time being, variables bound by λ are denoted by $x, y, z, \cdots$. Note that the infinite tree unwinding of the last recursion system is not a Böhm-tree, as it contains many β-redexes.

These systems of recursion equations allow us to express 'horizontal sharing' - *i.e.*, sharing as in a 'dag', as opposed to the 'vertical sharing' shown in the examples above. The following is an example of a system with horizontal sharing:

$$\alpha = \delta\delta, \ \delta = (\lambda x.\mathsf{F}(x))0 \ . \tag{1.1}$$

Since the right-hand side of the equations is restricted to λ-calculus terms, the horizontal sharing cannot appear inside a lambda-abstraction. This restricts the class of λ-graphs that we consider. For example, the graph of Figure 4 is not expressible. This limitation will be removed in the second part of the paper, Section 8, in which we introduce a framework with nested recursion equations. We restrict ourselves to systems without nesting since interesting observations can already be made.

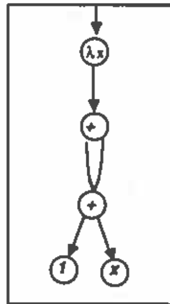


Figure 4: Lambda body with horizontal sharing.

Note that we admit in addition to pure λ-terms extended with recursion variables, operators from a first-order signature, like F and 0 above. We use a harmless mixture of applicative notation (with the application operator @ usually suppressed, except in pictures of λ-graphs) and 'functional' notation where operators have some arity (like the unary F above).

In presenting a recursion system, it is understood that the first (or topmost) equation is the leading equation, displaying the root of the λ-graph. When we want to be more precise, we will present the system displayed in (1.1) as

$$\langle \alpha \mid \alpha = \delta\delta, \ \delta = (\lambda x.\mathsf{F}(x))0 \rangle \ .$$

The order of the equations in the 'body' of the $\langle\ |\ \rangle$ construct is not important. Furthermore, we will consider recursion systems obtained from each other by 1-1 renaming of recursion variables, as identical. Thus,

$$\langle \delta \mid \delta = \gamma\gamma,\ \gamma = (\lambda x.\mathsf{F}(x))0\rangle$$

is the same expression as the previous one.

To summarize, until Section 8, we study systems of recursion equations of the form

$$\alpha_1 = M_1, \cdots, \alpha_n = M_n \ ,$$

where $M_1, \cdots, M_n$ are λ-calculus terms extended with constants, and the recursion variables $\alpha_1, \cdots, \alpha_n$ are distinct from each other.

1.1 Correspondence with graphs

It is straightforward to assign actual graphs to the recursion systems as introduced above. In the sequel there will be several examples. One feature should be mentioned explicitly: the nodes of the graph contain first-order operators (F), or application (@), or λx, or a variable $x, y, z, \cdots$. Other than that, a node may have a *name* $\alpha, \delta, \gamma, \cdots$. These correspond to the recursion variables in the recursion system. Note that also unnamed nodes may be present in the graph (corresponding to subterms in the system that have no name, like $x\alpha$ in $\langle \alpha \mid \alpha = \lambda x.x\alpha\rangle$). In the present setting, the root node of the λ-graph will always have a name.

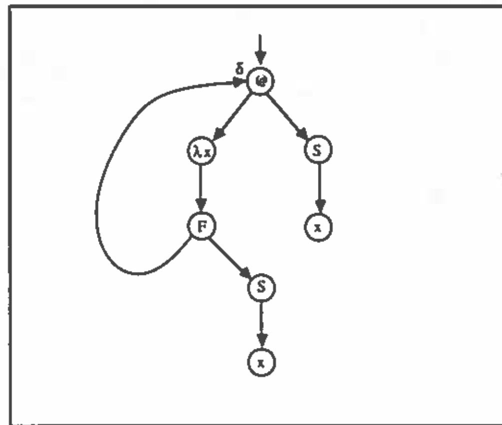


Figure 5: Cyclic lambda graph corresponding to $\delta = (\lambda x.\mathsf{F}(\delta, \mathsf{S}x))(\mathsf{S}x)$.

1.2 Free and bound variables

The notion of a variable $(x, y, \cdots)$ bound by a lambda follows from λ-calculus. For example, in the system

$$\alpha = (\lambda x.\mathsf{F}(\mathsf{G}x^2, \mathsf{S}x^3))\mathsf{S}x^1 \ ,$$

the variable x superscripted with 1 is free, and the $x's$ superscripted with 2 and 3 are bound. As another example, consider

$$\delta = (\lambda x.\mathsf{F}(\delta, \mathsf{S}x^2))(\mathsf{S}x^1) \ .$$

The x superscripted with 1 is free, while x^2 is considered to be bound. The above term is displayed in Figure 5. Our stipulation regarding free and bound variables points out a curious phenomenon; even though there is a path from the λx-node to the variable node x^1, x^1 is not bound by the λx node. We call this phenomenon *scope cut-off* (see Figure 6). This is consistent with other ways of presenting the cyclic λ-graph of Figure 5. For example, using the fixed point combinator Y, we would have $Y(\lambda\delta.(\lambda x.\mathsf{F}(\delta, \mathsf{S}x^2))(\mathsf{S}x^1))$, in which x^1 does indeed occur free.

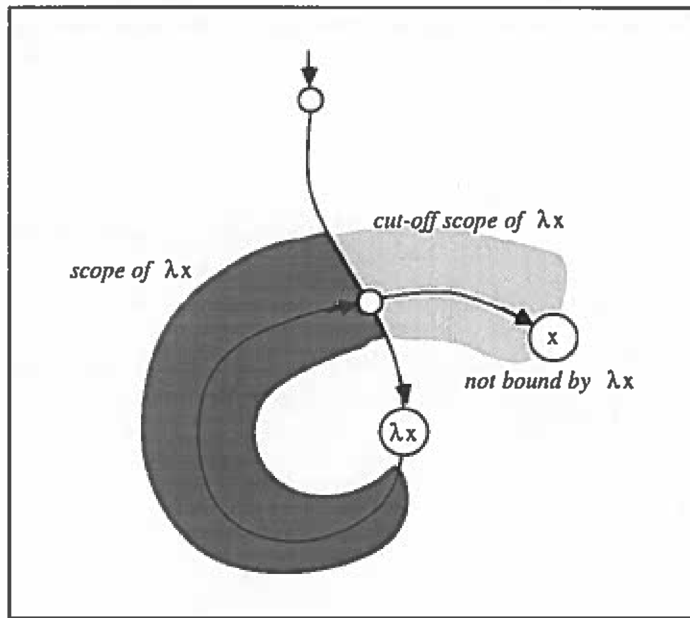


Figure 6: Scope cut-off phenomenon.

The same scope cut-off phenomenon occurs in the following system

$$\alpha = \lambda x.\delta,\ \delta = \mathsf{F}x\ ,$$

which is displayed in Figure 7; it is as if a name, in this case δ, stops the scope of a λ. As expected, this has some nasty consequences. With respect to the above system, substituting for δ in the first equation yields the system

$$\alpha = \lambda x.\mathsf{F}\underline{x},\ \delta = \mathsf{F}x\ ,$$

in which the underlined x has been captured. In order to avoid this free variable capture and still be able to use a naive version of substitution we adopt the convention that all free and bound variables have to be distinct from each other. Thus, we would express the term $\alpha = \lambda x.\delta,\ \delta = \mathsf{F}x$ as

$$\alpha = \lambda y.\delta,\ \delta = \mathsf{F}x\ .$$

2. Lambda graph rewriting

We now turn to the issue of defining β-reduction on λ-graphs or, equivalently, systems of recursion equations. Due to the possible presence of cycles, it may not immediately be clear

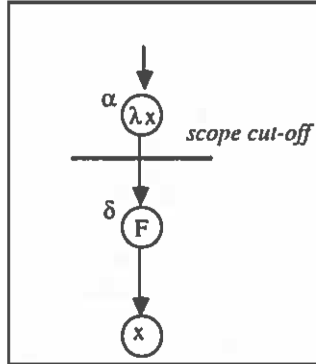


Figure 7: Cyclic lambda graph corresponding to $\alpha = \lambda x.\delta$, $\delta = \mathsf{F}x$.

what the 'right' notion of β-reduction is. In order to decide what is a right notion, we will compare, with respect to soundness, any notion of β-reduction for recursion systems with the infinitary version of the λ-calculus, as developed by Kennaway et al. [KKSdV95a]. First, we proceed in an intuitive fashion. We give some examples, where the redex being reduced is underlined:

$$\begin{array}{ll} \langle \alpha \mid \alpha = \underline{(\lambda x.\delta xx)\alpha}, \delta = (\lambda y.\alpha y)\delta \rangle & \longrightarrow_\beta \\ \langle \alpha \mid \alpha = \delta\alpha\alpha, \delta = \underline{(\lambda y.\alpha y)\delta} \rangle & \longrightarrow_\beta \\ \langle \alpha \mid \alpha = \delta\alpha\alpha, \delta = \alpha\delta \rangle \ . & \end{array}$$

Here, there is no problem. We call $(\lambda x.\delta xx)\alpha$ an *explicit* β-redex, since it is of the form $(\lambda x.M)N$. On the other hand, in a recursion system g, a subterm of the form αN is called an *implicit* β-redex if g contains an equation of the form $\alpha = \lambda x.M$. Examples of implicit β-redexes are $\delta(\mathsf{S}x)$ and $\alpha(\mathsf{S}y)$ in the example below:

$$\langle \alpha \mid \alpha = \lambda x.\delta(\mathsf{S}x), \delta = \lambda y.\alpha(\mathsf{S}y) \rangle \ .$$

An implicit redex αN must first be made explicit by substitution of $\lambda x.M$ for α, before it can be contracted (*i.e.*, β-reduced). The act of substitution will be denoted by $\longrightarrow_\mathsf{s}$; we will occasionally underline the variable we substitute for. Thus:

$$\begin{array}{ll} \langle \alpha \mid \alpha = \lambda x.\underline{\delta}(\mathsf{S}x), \delta = \lambda y.\alpha(\mathsf{S}y) \rangle & \longrightarrow_\mathsf{s} \\ \langle \alpha \mid \alpha = \lambda x.\underline{(\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x)}, \delta = \lambda y.\alpha(\mathsf{S}y) \rangle & \longrightarrow_\beta \\ \langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x)), \delta = \lambda y.\alpha(\mathsf{S}y) \rangle & \longrightarrow_\mathsf{gc} \\ \langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x)) \rangle \ . & \end{array}$$

In the last step, we have applied garbage collection (written as $\longrightarrow_\mathsf{gc}$) since the definition of δ is inaccessible from α.

Our stipulation that β-reduction can only be performed on explicit β-redexes in a system is a matter of choice; definitions of β-reduction directly on implicit β-redexes are possible. However, this stipulation makes it more clear, intuitively, what goes on. More importantly, making β-redexes explicit involves making a copy of part of the graph that is often necessary.

An example is:

$$\begin{array}{ll} \langle \alpha \mid \alpha = \mathsf{F}(\underline{\delta}0, \delta 1),\ \delta = \lambda x.x \rangle & \longrightarrow_{\mathsf{s}} \\ \langle \alpha \mid \alpha = \mathsf{F}(\underline{(\lambda x.x)0}, \delta(1)), \delta = \lambda x.x \rangle & \longrightarrow_{\beta} \\ \langle \alpha \mid \alpha = \mathsf{F}(0, \delta 1), \delta = \lambda x.x \rangle\ . & \end{array}$$

The substitution step has performed a copy of $\lambda x.x$, as is in this case anyway necessary.

2.1 The collapse problem

In orthogonal term graph rewriting (rewriting with an orthogonal first-order term rewriting system, admitting cyclic term graphs) and infinitary term rewriting (admitting infinite trees) it has been a matter of some discussion what to do with 'collapsing operators' such as a unary operator I with the rule $\mathsf{I}(x) \longrightarrow x$. Specifically, what should 'cyclic-I', that is, $\langle \alpha \mid \alpha = \mathsf{I}(\alpha) \rangle$, rewrite to? If this object rewrites to itself, then non-confluence arises. For, let J be another collapsing operator with $\mathsf{J}(x) \longrightarrow x$. Then $\langle \alpha \mid \alpha = \mathsf{I}(\mathsf{J}(\alpha)) \rangle$ rewrites to both $\langle \alpha \mid \alpha = \mathsf{I}(\alpha) \rangle$ and $\langle \alpha \mid \alpha = \mathsf{J}(\alpha) \rangle$. The simple solution is to proceed with rewriting; both of these last two expressions rewrite to $\langle \alpha \mid \alpha = \alpha \rangle$ which is perceived as a new object, that we will call $\bullet$ (black-hole), corresponding to the undefined μ-expression $\mu\alpha.\alpha$. We have used a new symbol to denote this result of a cyclic collapse, rather than use *e.g.*, $\perp$, as $\bullet$ represents a very specific and 'very undefined' kind of expression; it is a special case of expressions being undefined by lack of a head normal form. For a comparison of notions of undefinedness in orthogonal term (graph) rewriting see [AKK+94].

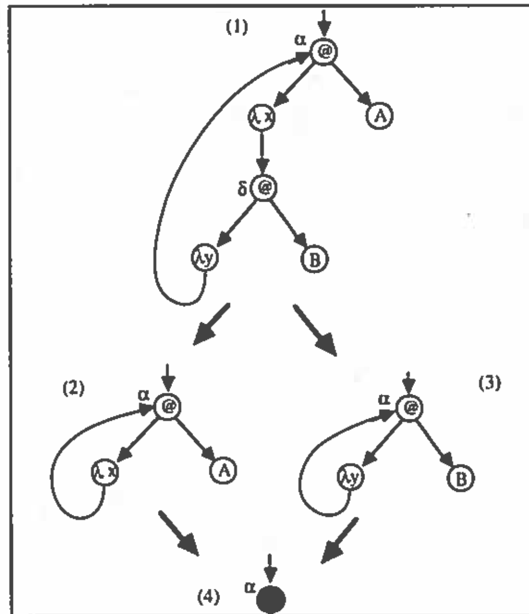


Figure 8: Reductions to black-hole.

Also in the present setting we need the introduction of the singularity point $\bullet$. *E.g.*, consider the λ-graph:

$$\alpha = \underbrace{(\lambda x.\delta)A}_{\rho}, \delta = \underbrace{(\lambda x.\alpha)B}_{\tau}.$$

β-rule :			
$(\lambda x.M)N$	$\longrightarrow_\beta$	$M[x := N]$	
Substitution :			
$\langle \alpha \mid \gamma = C[\delta], \delta = M, E\rangle$	$\longrightarrow_\mathsf{s}$	$\langle \alpha \mid \gamma = C[M], \delta = M, E\rangle$	
Black hole:			
$\langle \alpha \mid \gamma =_\circ \gamma, E\rangle$	$\longrightarrow_\bullet$	$\langle \alpha \mid \gamma = \bullet, E\rangle$	
Copying			
$\langle \alpha \mid E\rangle$	$\longrightarrow_\mathsf{c}$	$\langle \alpha' \mid F\rangle$	if ∃ a variable mapping σ, $\langle \alpha' \mid F\rangle^\sigma = \langle \alpha \mid E\rangle$
Naming :			
$\langle \alpha \mid \gamma = C[M], E\rangle$	$\longrightarrow_\mathsf{n}$	$\langle \alpha \mid \gamma = C[\delta], \delta = M, E\rangle$	if the free variables of M do not occur bound in $C[\square]$ and M is not a variable
Garbage collection:			
$\langle \alpha \mid E, F\rangle$	$\longrightarrow_\mathsf{gc}$	$\langle \alpha \mid E\rangle$	if F is non-empty and orthogonal to E and α

Table 1: Reduction rules of λΘ.

Contracting the ρ-redex yields

$$\alpha = \delta,\ \delta = (\lambda x.\alpha)B$$

which is equivalent to

$$\alpha = (\lambda x.\alpha)B\ .$$

Contracting the τ-redex yields

$$\alpha = (\lambda x.\delta)A,\ \delta = \alpha$$

which is equivalent to

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

Both contracted graphs yield after one more reduction $\alpha = \alpha$, and this is equivalent to $\alpha = \bullet$. (See Figure 8). Note that also mutual vacuous dependencies of recursion variables have to be replaced by $\bullet$. *E.g.*, $\langle \alpha \mid \alpha = \delta, \delta = \alpha\rangle \twoheadrightarrow \bullet$. Or, inside a system:

$$\langle \alpha \mid \alpha = \mathsf{F}((\lambda x.x)\delta), \delta = (\lambda y.y)\delta\rangle \twoheadrightarrow \langle \alpha \mid \alpha = \mathsf{F}(\delta), \delta = \delta\rangle \longrightarrow \langle \alpha \mid \alpha = \mathsf{F}(\bullet)\rangle\ .$$

3. The λΘ-calculus

The naive way of reducing possibly cyclic redexes introduced so far, which we call the λΘ-calculus, is given in Table 1. Notation: The metavariables E, F range over unordered sequences (possibly empty) of recursion equations. $M[x := N]$ denotes the substitution of N for each free occurrence of x in M. $C[\square]$ represents a λ-calculus context. The binding $\gamma =_\circ \gamma$ in the *Black hole* rule stands for the sequence of bindings $\gamma = \gamma_1, \cdots, \gamma_n = \gamma$. F orthogonal to a sequence of equations E or to a variable α means that the bound recursion variables of

F do not intersect with the free variables of E and α. In the *Substitution* rule, the equations $\gamma = C[\delta]$ and $\delta = M$ can overlap as in the following substitution step:

$$\langle \alpha \mid \alpha = \lambda x.x\alpha \rangle \longrightarrow_{\mathsf{s}} \langle \alpha \mid \alpha = \lambda x.x(\lambda x.x\alpha) \rangle \ ,$$

in which both δ and γ are mapped into α. The operation of copying differs from substitution in the sense that copying never gets rid off recursion variables. Given two recursion systems g and g_1, g copies to g_1 if there exists a mapping σ from recursion variables to recursion variables (which is extended in the usual way to a system of recursion equations) such that $g_1^\sigma = g$, leaving the free recursion variables of g_1 unchanged. For example,

$$\langle \alpha \mid \alpha = \mathsf{F}(\gamma), \gamma = \mathsf{G}(\alpha) \rangle \longrightarrow_{\mathsf{c}} \langle \alpha \mid \alpha = \mathsf{F}(\gamma), \gamma = \mathsf{G}(\alpha'), \alpha' = \mathsf{F}(\gamma'), \gamma' = \mathsf{G}(\alpha') \rangle \ ,$$

where the variable mapping σ is: α, α' are mapped to α, and γ, γ' are mapped to γ. (See [AK95] for a thorough discussion of copying and its properties.) The proviso for the operation of naming, which is written as $\longrightarrow_{\mathsf{n}}$, is to forbid reductions of the form

$$\langle \alpha \mid \alpha = \lambda x.\mathsf{F}x \rangle \longrightarrow \langle \alpha \mid \alpha = \lambda x.\delta, \delta = \mathsf{F}\underline{x} \rangle \ ,$$

in which the underlined x gets out of scope.

To understand why we admit, in addition to substitution, also the operations of copying and naming, we make an excursion into the first-order case. Substitution by itself causes already non-confluence in the first-order case. For, consider the recursion system without any rewrite rule:

$$\langle \alpha \mid \alpha = \mathsf{S}(\delta), \delta = \mathsf{S}(\alpha) \rangle \ .$$

By substitution and garbage collection this expression yields on the one hand

$$\langle \alpha \mid \alpha = \mathsf{S}(\mathsf{S}(\alpha)) \rangle \tag{3.2}$$

on the other hand

$$\langle \alpha \mid \alpha = \mathsf{S}(\delta), \delta = \mathsf{S}(\mathsf{S}(\delta)) \rangle \ . \tag{3.3}$$

These two results cannot be made convergent by further substitutions; at each point in time system (3.2) will have an even number of S's, while system (3.3) will contain an odd number of S's. However, by allowing re-introduction of names (*i.e., Naming*) we can restore

$$\langle \alpha \mid \alpha = \mathsf{S}(\mathsf{S}(\alpha)) \rangle$$

to

$$\langle \alpha \mid \alpha = \mathsf{S}(\delta), \delta = \mathsf{S}(\alpha) \rangle$$

and converge again. As shown in [AK95], confluence of substitution and naming is guaranteed if the system contains also the operation of copying. Thus, in analogy with the first-order case, we consider next to substitution also the operations of naming and copying, hoping to prove confluence of $\lambda\Theta$. However, as shown in the next section, there are some nasty surprises.

Remark 3.1 It is interesting to observe that *Naming* can cause a non-terminating computation to terminate. *E.g.*,

$$\alpha = \lambda y.\alpha 0 \twoheadrightarrow \alpha = \lambda y.\alpha 0 \twoheadrightarrow \alpha = \lambda y.\alpha 0 \twoheadrightarrow \cdots \; .$$

Since $\alpha 0$ does not depend on the bound variable y it can be given a name. Then:

$$\alpha = \lambda y.\alpha 0 \longrightarrow_{\mathsf{n}} \begin{array}{l}\alpha = \lambda y.\delta, \\ \delta = \alpha 0\end{array} \longrightarrow_{\mathsf{s}} \begin{array}{l}\alpha = \lambda y.\delta, \\ \delta = (\lambda y.\delta)0\end{array} \longrightarrow_{\beta} \begin{array}{l}\alpha = \lambda y.\delta, \\ \delta = \delta\end{array} \twoheadrightarrow \alpha = \lambda y.\bullet \; .$$

The above term $\alpha = \lambda y.\alpha 0$ can be seen as an infinite tower of collapsing contexts. As will be discussed in Section 5, this constitutes a source of non-confluence in the infinitary calculi. To solve the matter, in the infinitary setting such terms are reduced to $\lambda y.\bullet$.

This example points out that an approach to cyclic objects based on labelled nodes and edges is unsuitable to describe common program manipulations, as the one described above. In other words, the graph should be given some structure which delimits the body of a lambda, as naming a node does in this simple framework.

4. A counterexample to confluence of $\lambda\Theta$

Consider the reductions (displayed in Figure 9) :

$$\begin{array}{lcl}
\begin{array}{l}\alpha = \lambda x.\delta(\mathsf{S}x), \\ \delta = \lambda y.\alpha(\mathsf{S}y)\end{array} & \longrightarrow_{\mathsf{s}} \begin{array}{l}\alpha = \lambda x.(\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x), \\ \delta = \lambda y.\alpha(\mathsf{S}y)\end{array} \longrightarrow_{\beta} & \begin{array}{l}\alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x)), \\ \delta = \lambda y.\alpha(\mathsf{S}y)\end{array} \\
\downarrow \mathsf{s} & & \vdots \\
\begin{array}{l}\alpha = \lambda x.\delta(\mathsf{S}x), \\ \delta = \lambda y.(\lambda x.\delta(\mathsf{S}x))(\mathsf{S}y)\end{array} & & \downarrow \\
\downarrow \beta & & \\
\begin{array}{l}\alpha = \lambda x.\delta(\mathsf{S}x), \\ \delta = \lambda y.\delta(\mathsf{S}(\mathsf{S}y))\end{array} & \dashrightarrow & ?
\end{array}$$

By using the same parity argument as in the previous section one can see that the two systems obtained are clearly *out-of-synch*. The situation is even more serious, less curable than in the first-order case since also the operations of naming and copying do not help. The two expressions

$$\alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x)) \qquad \text{and} \qquad \alpha = \lambda x.\delta(\mathsf{S}x), \delta = \lambda y.\delta(\mathsf{S}(\mathsf{S}y))$$

are irreversibly separated with respect to any set of operations on λ-graphs that is 'sound' in a sense that we will elaborate in the next section.

The above counterexample corresponds to unfolding or inlining the redexes ρ_1 and ρ_2, respectively, in the following mutually recursive definitions of CAML:

$$\begin{array}{l}
\# \text{ let rec odd = fun x} \rightarrow \text{if x = 0 then false else } \underbrace{\text{even}(x-1)}_{\rho_1} \\
\quad \text{and even = fun x} \rightarrow \text{if x = 0 then true else } \underbrace{\text{odd}(x-1)}_{\rho_2} \;;;
\end{array}$$

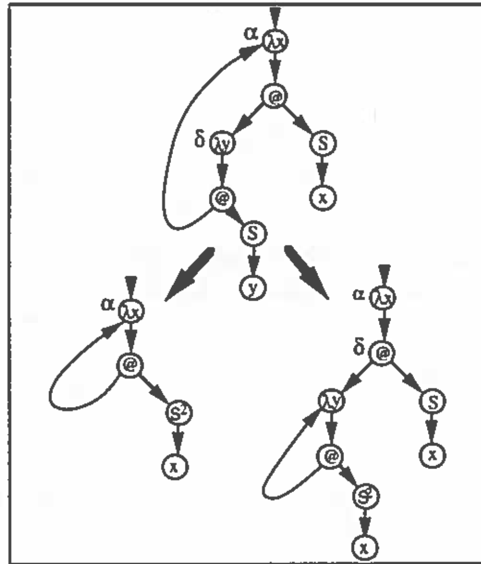


Figure 9: Failure of confluence after reduction of spine-cyclic redexes.

The absence of a common reduct means that depending on how we apply these transformations we get different programs, which, even though they might produce the same observable result, are different from an intensional point of view. As an example, unfolding ρ_1 first triggers the application of the *unused lambda expressions* transformation [App92], and thus getting rid off the definition of even.

Analysis of the counterexample
The above counterexample not only is a counterexample to confluence, but even to weak confluence. For ordinary λ-calculus, weak confluence is simple to prove by an inspection of 'elementary reduction diagrams'. Typical for these elementary reduction diagrams is that in the converging sides, one has to contract the descendants (residuals) of the redexes contracted in the diverging sides. So what goes wrong in the present case, when we try to prove weak confluence? Let us review the counterexample:

$$\alpha = \lambda x.[\delta(\mathsf{S}x)]_1, \delta = \lambda y.[\alpha(\mathsf{S}y)]_2 \ ,$$

where we have indicated the two redexes, 1 and 2, that play a role. Both are implicit redexes. Reduction of redex 1 requires making it explicit:

$$\alpha = \lambda x.[(\lambda y.[\alpha(\mathsf{S}y)]_2)(\mathsf{S}x)]_1, \delta = \lambda y.[\alpha(\mathsf{S}y)]_2 \ .$$

Garbage collection yields: $\alpha = \lambda x.[(\lambda y.[\alpha(\mathsf{S}y)]_2)(\mathsf{S}x)]_1$. The redex marked 1 can now be contracted, with result (S^2 stands for $\mathsf{S}(\mathsf{S}(x))$):

$$\alpha = \lambda x.[\alpha(\mathsf{S}^2 x)]_2 \ .$$

In the other direction, we contract redex 2, after explicitation:

$$\alpha = \lambda x.[\delta(\mathsf{S}x)]_1, \ \delta = \lambda y.[(\lambda x.[\delta(\mathsf{S}x)]_1)(\mathsf{S}y)]_2 \ .$$

Contraction of the redex 2 yields:

$$\alpha = \lambda x.[\delta(\mathsf{S}x)]_1,\ \delta = \lambda y.[\delta(\mathsf{S}^2 y)]_1\ .$$

So, in analogy of pure λ-calculus, we would expect that all we have to do is: complete the following elementary reduction diagram, by contraction of the respective residuals.

$$\begin{array}{ccc} \alpha = \lambda x.[\delta(\mathsf{S}x)]_1, \delta = \lambda y.[\alpha(\mathsf{S}y)]_2 & \rightarrow_1 & \alpha = \lambda x.[\alpha(\mathsf{S}^2x)]_2 \\ \downarrow_2 & & \downarrow_2 \\ \alpha = \lambda x.[\delta(\mathsf{S}x)]_1, \delta = \lambda y.[\delta(\mathsf{S}^2y)]_1 & \rightarrow_1 & ? \end{array}$$

Now the reason of the failure of confluence comes to the surface: reduction of redexes 1 in $\alpha = \lambda x[\delta(\mathsf{S}x)]_1, \delta = \lambda y.[\delta(\mathsf{S}^2y)]_1$, or rather a complete development of the set of 1-redexes, is not possible. Likewise a complete development of the singleton set of 2-redexes in

$$\alpha = \lambda x[\alpha(\mathsf{S}^2x)]_2\ ,$$

is not possible. We will show this for the latter case, the 2-redex; the other case of the 1-redex is similar.

Let us try to completely develop the singleton set of 2-redexes in $\alpha = \lambda x.[\alpha(\mathsf{S}^2x)]_2$. This requires first of all, making the implicit 2-redex explicit. Therefore we substitute for α, result:

$$\alpha = \lambda x.[(\lambda x.[\alpha(\mathsf{S}^2x)]_2)\mathsf{S}^2x]_2\ .$$

Now the original 2-redex has indeed become explicit, but in doing so we have created another implicit 2-redex. Making this in turn explicit does not help:

$$\alpha = \lambda x.[(\lambda x.[(\lambda x.[\alpha(\mathsf{S}^2x)]_2)\mathsf{S}^2x]_2)\mathsf{S}^2x]_2\ .$$

So the reason of the fact that the diagram cannot be completed, is that the sets of residuals cannot be made explicit, cannot be pre-developed. Hence they cannot be developed.

Another Analysis of the counterexample

Consider the following Abstract Reduction System, with as elements: singleton sets of natural numbers n, pairs of natural numbers (n,m), and alternative pairs of natural numbers $[n,m]$. There are the following reduction rules:

$$\begin{array}{lcl} \{n\} & \longrightarrow & \{2n\} \\ \{n\} & \longrightarrow & (n,n) \\ \{n\} & \longrightarrow & [n,n] \\ (n,m) & \longrightarrow & (n+m,m) \\ (n,m) & \longrightarrow & (n,2m) \\ [n,m] & \longrightarrow & \{n+m\} \\ [n,m] & \longrightarrow & (n,n+m)\ . \end{array}$$

We claim that these are not confluent. Proof: $[1,1] \longrightarrow \{2\}$ and $[1,1] \longrightarrow (1,2)$. Any reduct of $\{2\}$ is of the form $\{e\}$ or (e_1, e_2) or $[e_1, e_2]$ with e, e_1, e_2 even. Any reduct of $(1,2)$ is of the form (o, e) with o odd and e even.

Using this abstract non-confluent fact, we can give a sketch of the non-confluence of reductions of the system:

$$\begin{cases} \alpha = \lambda x.\delta(\mathsf{S}(x)) \\ \delta = \lambda y.\alpha(\mathsf{S}(y)) \end{cases}$$

Let us abbreviate:

$$\begin{array}{lll} \{n\} & : & \alpha = \lambda x.\alpha(\mathsf{S}^n x) \\ {[n,m]} & : & \begin{cases} \alpha = \lambda x.\delta(\mathsf{S}^n(x)) \\ \delta = \lambda y.\alpha(\mathsf{S}^m(y)) \end{cases} \\ (n,m) & : & \begin{cases} \alpha = \lambda x.\delta(\mathsf{S}^n(x)) \\ \delta = \lambda y.\delta(\mathsf{S}^m(y)) \end{cases} \end{array}$$

Then indeed the abstract rewrite rules above are obtained by β-reduction on systems of equations together with a limited form of copying. Hence the original system, which in abbreviation is $[1,1]$, is not confluent. Actually this 'proof' is only giving the basic idea, it is not complete since *e.g.*, the system abbreviated as $\{2\}$ gives rise by copying to other systems than the ones above. For example:

$$\{2\} \longrightarrow_{\mathsf{c}} \begin{cases} \alpha = \lambda x.\delta(\mathsf{S}^2(x)) \\ \delta = \lambda y.\gamma(\mathsf{S}^2(y)) \\ \gamma = \lambda x.\delta(\mathsf{S}^2(x)) \end{cases}$$

But also now, all S's ever appearing in reducts/expansions of the latter system will have even exponents. On the other hand, the system $(1,2)$ can be expanded *e.g.*, as follows:

$$(1,2) \longrightarrow_{\mathsf{c}} \begin{cases} \alpha = \lambda x.\delta(\mathsf{S}(x)) \\ \delta = \lambda y.\gamma(\mathsf{S}^2(y)) \\ \gamma = \lambda x.\delta(\mathsf{S}^2(x)) \end{cases}$$

And now in all reducts/expansions of the latter system, the S in the equation for α will have odd exponent, and the S's in all the other equations will have even exponents.

This phenomenon may be thought to be dependent of our particular choice of reduction for cyclic redexes, consisting of a substitution step followed by a familiar β-step. However, we claim that it is robust, in fact, as we are going to explain in the next section, the same phenomenon occurs in the infinitary version of λ-calculus [KKSdV95a].

5. Infinitary lambda calculus

As semantics of λ-graph rewriting we take the infinitary λ-calculus, as introduced by Kennaway et al. [KKSdV95a]. The infinitary λ-calculus provides us with a notion of correctness of proposed definitions of β-reduction of λ-graphs, and explains the counterexamples for (finitary) confluence of λ-reduction of such graphs. In this section we will give a short exposition of some of the concepts introduced in [KKSdV95a, KKSdV95b].

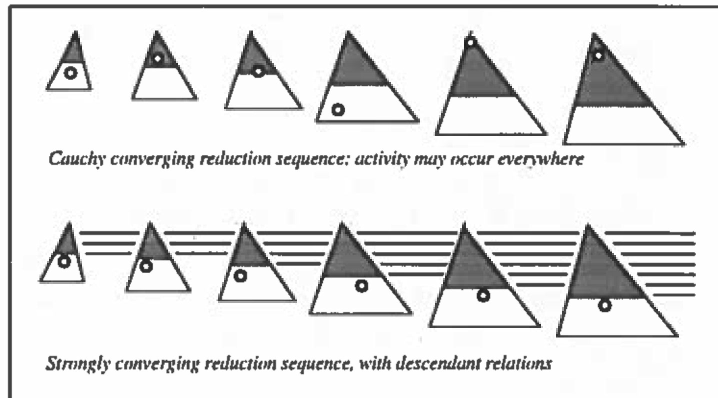


Figure 10: Converging and strongly converging reduction sequences.

We first underline the difference between convergent and strongly convergent reductions. In short, a strongly convergent reduction is such that the stable prefix of the term is increasing (see Figure 10), that is, the depth of contracted redexes tends to infinity. There is an analogy between the finitary and infinitary settings; a finite reduction corresponds to a strongly convergent reduction, while an infinite reduction corresponds to a non strongly convergent reduction.

In transfinite orthogonal term rewriting there is a single source of failure of infinitary confluence: the presence of collapsing operators, such as the I or K combinators, enabling one to build trees that consist of an infinite tower of collapsing operators, or rather collapsing contexts. This is proved in [KKSdV95b]. In the infinitary λ-calculus, that is also a source of non-confluence. However, the matter is more complicated, there is another phenomenon that causes infinitary non-confluence, not due to collapsing contexts. To explain this, we first need the concepts of development and complete development, which are a generalization of the classical notions of λ-calculus.

Definition 5.1 Let M be a λ-tree, and S be a possibly infinite set of redexes in M.
(i) A *development* of S is a reduction, possibly infinite, in which only descendants of members of S are contracted.
(ii) A *complete development* of S is a development which is strongly convergent and after which no descendant of a redex of S is left.

A classical lemma in λ-calculus is the Finite Developments Lemma, stating that any development must terminate (see Barendregt [Bar84]). Of course, we cannot have that for the infinitary λ-calculus, since it admits infinitely many redexes to be developed. But there is an analogous statement, bearing in mind the analogy that we hinted at before between finite λ-reductions and strongly converging infinitary reductions on the one hand, and infinite λ-reductions and diverging infinitary reductions on the other hand. This would be that any development strongly converges. This is however not the case, and this gives rise to a failure of infinitary confluence, as shown in the next example.

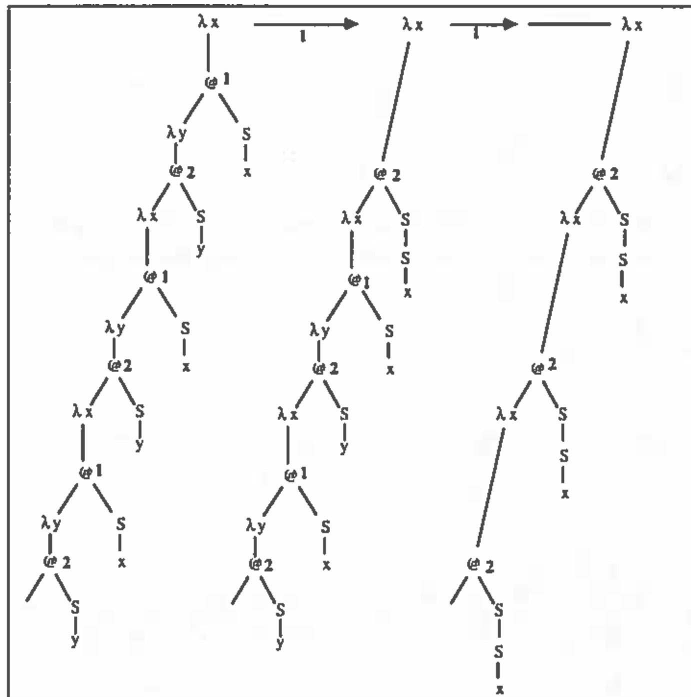


Figure 11: Complete development of the redexes marked as 1.

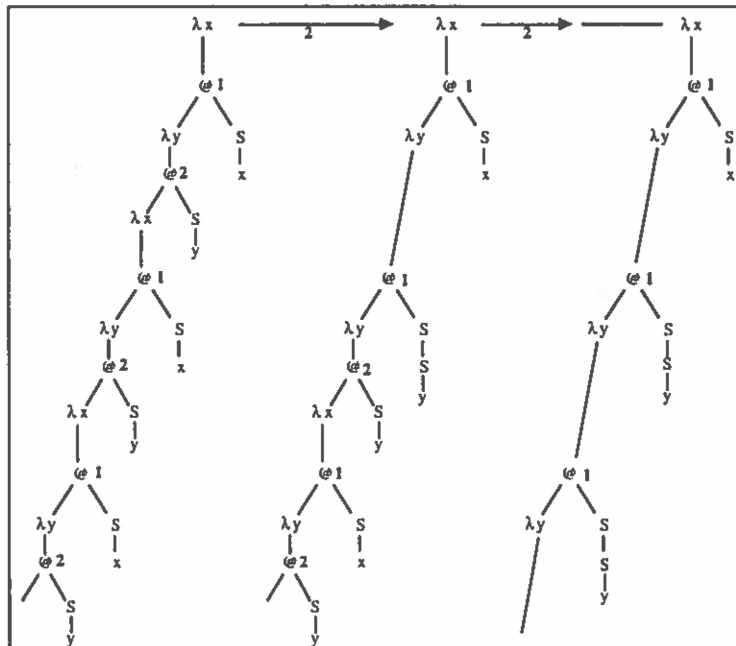


Figure 12: Complete development of the redexes marked as 2.

Consider the infinite unwinding of the term

$$\langle \alpha \mid \alpha = \lambda x.\delta(\mathsf{S}x), \delta = \lambda y.\alpha(\mathsf{S}y) \rangle \ ,$$

which, as was discussed in the previous section, was leading to two non-converging reductions. Let S_1 and S_2 be the two sets of redexes descending from the two redexes $\alpha(\mathsf{S}y)$, $\delta(\mathsf{S}x)$ in that infinite term. The result of the complete development of S_1 and S_2 is shown in Figures 11 and 12, respectively. The two infinite terms so obtained do not have a common reduct. We present a stylized version of the proof, for a formal exposition the reader can consult [KKSdV95a]. Consider the TRS with for every $n \geq 0$, a unary operator $\underline{n}$. There are infinitely many rewrite rules:

$$\underline{n}(\underline{m}(x)) \longrightarrow (\underline{n+m})(x) \ .$$

This is a confluent and terminating TRS. It is not orthogonal. The infinitary version is not confluent. For, consider the following infinite terms (where we have omitted brackets in the convention of association to the right):

$$\begin{array}{cccccccccccll}
\underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots & \longrightarrow \\
 & 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots & \longrightarrow \\
 & 2 & & 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots & \longrightarrow_{\omega} \\
 & 2 & & 2 & & 2 & & 2 & & 2 & & \cdots &
\end{array}$$

where we have 'developed' the underlined redexes, that is, the redexes at even positions. This corresponds to the reduction of redexes marked with **1** in Figure 11, whose leftmost infinite tree is represented as the infinite term $111\cdots$ (*i.e.*, at each level the tree contains one symbol S). The complete development of the redexes marked with **2** in Figure 12 corresponds to the reduction of the redexes at odd positions, yielding:

$$\begin{array}{cccccccccccll}
1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \cdots & \longrightarrow \\
1 & & 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \cdots & \longrightarrow \\
1 & & 2 & & 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \cdots & \longrightarrow_{\omega} \\
1 & & 2 & & 2 & & 2 & & 2 & & 2 & \cdots &
\end{array}$$

Now it is clear that the two infinite terms $2222\cdots$ and $12222\cdots$ have no common reduct. A side result of this example is that for confluent and terminating TRSs the generalization to infinitary rewriting does not work out well; apparently the orthogonality condition is needed.

Identifying the larger class of terms without head normal form does restore confluence for the infinitary λ-calculus. A term M has a head normal form if it reduces to some term of the form $\lambda x_1 \cdots x_n.yN_1 \cdots N_m$, $(n, m \geq 0)$.

Theorem 5.2 *The infinitary lambda calculus extended with the rule:*

$$M \rightarrow \Omega \ \textit{if } M \textit{ has no head normal form}$$

is infinitary confluent.

Proof: See [KKSdV95a]. □

Remarkably, this is not the case in λ-graph rewriting. Before presenting the counterexample, we discuss our criterion of soundness.

Soundness
Using the infinitary λ-calculus we can now formulate a soundness criterion for transformations of $\lambda\Theta$-expressions. (Also the various transformations in Section 8 satisfy this criterion.)

Definition 5.3 Let g, g' be two recursion systems. Let $T(g)$ be the tree unwinding of g. We will say that a transformation $g \longrightarrow g'$ is *sound* (with respect to the infinitary λ-calculus) if $T(g) \longrightarrow_{\leq\omega} T(g')$. (Here $\longrightarrow_{\leq\omega}$ denotes possibly infinitary reduction, that is a sequence of ω or less (possibly 0) β-steps.)

The proof of soundness of the $\lambda\Theta$-calculus will be given after having introduced a notion of substitution that captures the tree unwinding of a recursion system.

6. Regular developments and another counterexample

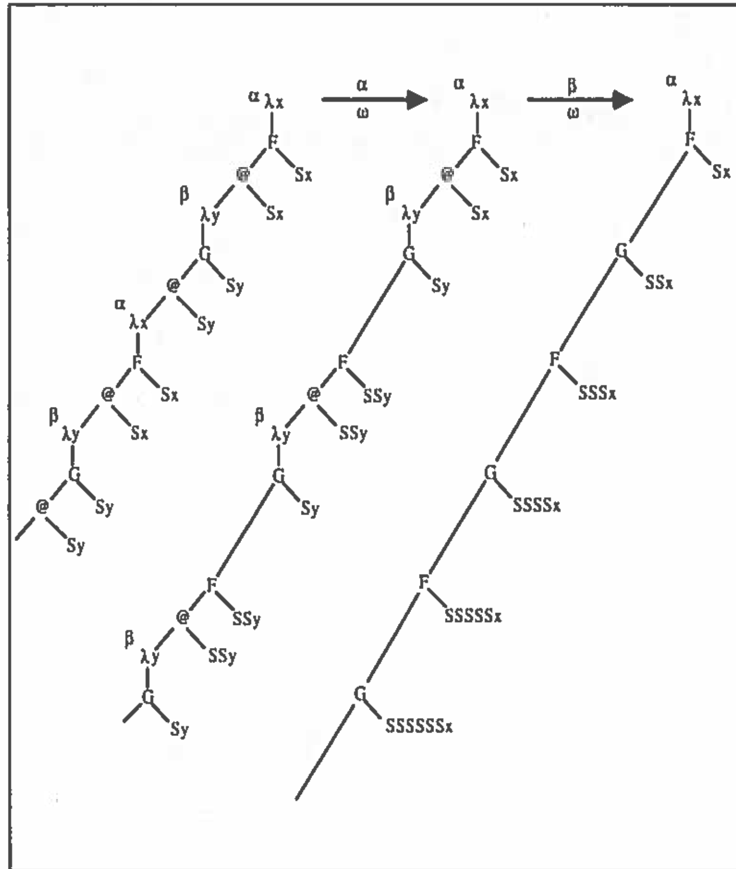


Figure 13: Infinite reduction yielding a non-regular tree.

It may be thought that non-confluence in the $\lambda\Theta$-calculus only arises because of expressions that after unwinding to the corresponding infinite λ-tree have no head normal form. Or, equivalently, that confluence can be restored by equating all $\lambda\Theta$-expressions that have no head

normal form as in the infinitary λ-calculus. However, this is not the case: *non-confluence in $\lambda\Theta$ also may arise for expressions that have an infinitary normal form.*

Before presenting the example, we remind the reader that a λ-tree is regular if it contains modulo isomorphism only finitely many different sub-trees. A development is regular, if it is a development of a set of redexes in a regular λ-tree, and the result of the development exists and yields again a regular λ-tree. Consider:

$$\langle \alpha \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\ \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y) \rangle \ .$$

Unwinding these recursion equations yields the infinite λ-tree in the leftmost corner of Figure 13. A development of the redexes with function part α yields a regular tree, as in the figure. Likewise a development of the redexes with function part γ yields a regular tree. But developing both sets of redexes yields a non-regular tree, namely, the rightmost one of Figure 13. Hence, we have another counterexample to confluence of $\lambda\Theta$ analogous to the counterexample in Section 4. This can be seen as follows: the first development of redexes with function part α, corresponds in $\lambda\Theta$ to β-reductions of $\alpha(\mathsf{S}y)$, after substituting for α to make the redex explicit. Likewise, the second development of redexes with function part γ corresponds to reducing $\gamma(\mathsf{S}x)$. Due to the soundness of $\lambda\Theta$, the common reduct in $\lambda\Theta$ must give rise after unwinding to a regular tree, and not to the non-regular tree resulting from executing both developments.

Summarizing, we have the situations as in Figure 14, where the lower plane is that of λ-trees and their infinite reductions, and the upper plane is that of λ-graphs and their finite reductions. The planes are related by tree unwinding. Figure 14(*i*) displays the 'normal' situation. Figure 14(*ii*) refers to the counterexample in Section 4, that is, the loss of confluence in both λ-graph rewriting and in the infinitary λ-calculus. Figure 14(*iii*) refers to the reduction of systems such as the following:

$$\langle \alpha \mid \alpha = \lambda x.\mathsf{F}(\gamma x), \gamma = \lambda y.\mathsf{G}(\alpha y) \rangle \ .$$

In $\lambda\Theta$ we obtain the same out-of-synch phenomenon as in Section 4. Instead, the corresponding infinite term reduces to the infinitary normal form $\lambda x.(\mathsf{FG})^{\omega}$, independently of the order of reduction of the redexes of the form γx and αy. Figure 14(*iv*) refers to the counterexample presented in this section.

7. Notions of substitution

A way of avoiding the loss of confluence would be to forbid reduction of redexes whose root is lying on a cycle. But that is hardly satisfactory. (See the Fibonacci graph displayed in Figure 2.) Clearly, due to its computational content, we have an intuition of such a graph as unproblematic, and not leading to non-confluence. Going back to the analysis of the first counterexample, it is not hard to see what causes a set of redexes in a recursion system to resist a complete pre-development. This occurs only if there is a cyclic configuration in the

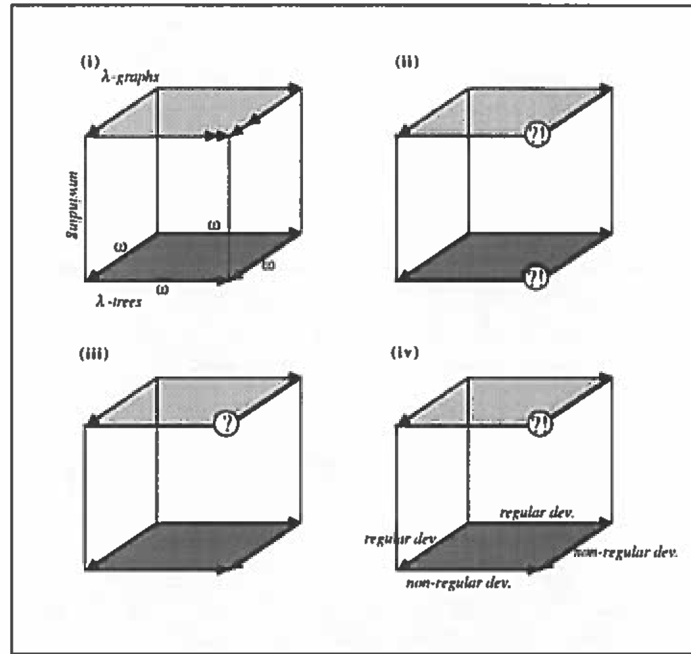


Figure 14: Relation between λ-graph and λ-tree reductions.

system as follows:

$$\begin{aligned}\alpha_0 &= \lambda x_1.C_1[(\alpha_1 M_1)]\\ &\dots\\ \alpha_i &= \lambda x_{i+1}.C_{i+1}[(\alpha_{i+1} M_{i+1})]\\ &\dots\\ \alpha_n &= \lambda x_{n+1}.C_{n+1}[(\alpha_0 M_0)]\ .\end{aligned}$$

Here the $\alpha_i M_i$ are the implicit β-redexes that we want to pre-develop. If we underline all the α_i in the above system and apply substitution, it so happens that those underlines can never disappear. This suggests looking for a new form of substitution that leads to finite developments.

The new substitution, called *acyclic substitution* (written as $\longrightarrow_{\mathsf{as}}$), consists of defining an order on the nodes of a graph, or equivalently on the recursion variables (see Figure 15), and then allowing substitution upwards only. More precisely: call two nodes *cyclically equivalent* if they are lying on a common cycle. A *plane* is a cyclic equivalence class. If there is a path from node s to node t, and s, t are not in the same plane, we define $s > t$. Let γ be the name associated to node s, and δ the name associated to node t, then $\gamma > \delta$. Acyclic substitution is then defined as follows:

$$\langle \alpha \mid \gamma = C[\delta], \delta = M, E\rangle \longrightarrow_{\mathsf{as}} \langle \alpha \mid \gamma = C[M], \delta = M, E\rangle \qquad \text{if } \gamma > \delta\ .$$

In $C[\delta]$ just one occurrence of δ is displayed and replaced by M. So in Figure 15, displaying the system

$$\langle \alpha \mid \alpha = \mathsf{F}(\nu, \delta, \alpha), \nu = \mathsf{H}(\mathsf{G}(\nu), \gamma), \gamma = \mathsf{H}(\mathsf{C}, \delta), \delta = \mathsf{G}(\gamma)\rangle$$

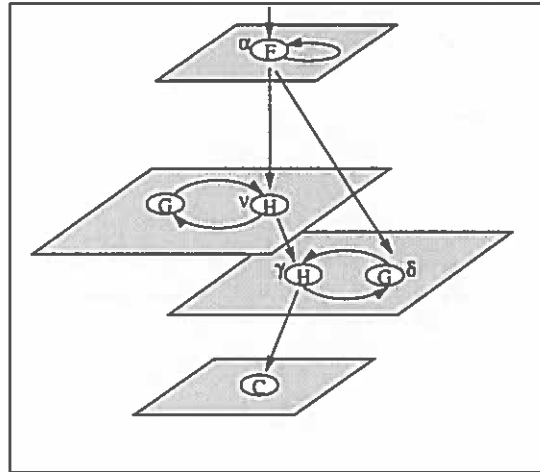


Figure 15: Ordering among recursion variables.

β-rule :			
$(\lambda x.M)N$	$\longrightarrow_{\beta}$	$M[x := N]$	
Acyclic substitution :			
$\langle \alpha \mid \gamma = C[\delta], \delta = M, E\rangle$	$\longrightarrow_{\mathsf{as}}$	$\langle \alpha \mid \gamma = C[M], \delta = M, E\rangle$	if $\gamma > \delta$

Table 2: Reduction rules of $\lambda\Phi$.

the only $\longrightarrow_{\mathsf{as}}$ -steps are from δ in α, from ν in α, from γ in ν. The new calculus, called $\lambda\Phi$, that embodies acyclic substitution is given in Table 2.

Fact 7.1 *Acyclic substitution is non terminating.* E.g.,

$$\alpha = \mathsf{F}\gamma, \gamma = \mathsf{G}\gamma \longrightarrow \alpha = \mathsf{FG}\gamma, \gamma = \mathsf{G}\gamma \longrightarrow \alpha = \mathsf{FGG}\gamma, \gamma = \mathsf{G}\gamma \longrightarrow \cdots \quad .$$

Referring to the above reduction note that the second step involves the reduction of a new redex. If reduction is restricted to "old" redexes only then acyclic substitution becomes terminating. To that end, let us introduce an underlined substitution calculus which we call $\underline{\mathsf{As}}$. The terms of the new calculus are systems of recursion equations with underlined recursion variables, with the proviso that the underlined variables have to belong to an acyclic substitution redex. For example, the system

$$\alpha = \mathsf{F}\underline{\gamma}, \gamma = \mathsf{G}\gamma \ ,$$

is a legal term. On the other hand, the system

$$\alpha = \mathsf{F}\gamma, \gamma = \mathsf{G}\underline{\gamma}$$

is not legal since $\gamma \not> \gamma$. The rule of $\underline{\mathsf{As}}$ is:

$$\langle \alpha \mid \gamma = C[\underline{\delta}], \delta = M, E\rangle \longrightarrow_{\underline{\mathsf{as}}} \langle \alpha \mid \gamma = C[M], \delta = M, E\rangle \ .$$

From now on, we will identify an acyclic substitution redex with the variable we are substituting for. *E.g.*, given the system $\alpha = \mathsf{F}\underline{\gamma}, \gamma = 0$, we will say that $\underline{\gamma}$ is a redex.

Lemma 7.2 *Let $g \longrightarrow_{\underline{\mathsf{as}}} g_1$ by reducing redex $\underline{\delta}$ and $g \longrightarrow_{\underline{\mathsf{as}}} g_2$ by reducing redex $\underline{\gamma}$, then a common reduct g_3 can be found by reducing in g_1 all descendants of $\underline{\gamma}$ and in g_2 all descendants of $\underline{\delta}$.*

Proof: Let $g \longrightarrow_{\underline{\mathsf{as}}} g_1$ by substituting for $\underline{\delta}$ in α, and $g \longrightarrow_{\underline{\mathsf{as}}} g_2$ by substituting for $\underline{\gamma}$ in η. The only interesting case is when $\delta = \eta$ or $\gamma = \alpha$. In other words, the two substitutions have a cyclic plane in common (see Figure 15). Let us assume $\delta = \eta$. We have:

$$
\begin{array}{rlcrl}
g \equiv & \alpha = C[\underline{\delta}], & \xrightarrow[\delta]{} & g_1 \equiv & \alpha = C[C_1[\underline{\gamma}]], \\
 & \delta = C_1[\underline{\gamma}], & & & \delta = C_1[\underline{\gamma}], \\
 & \gamma = N & & & \gamma = N \\
 & \Big\downarrow \gamma & & & \downarrow \gamma \\
 & & & & \alpha = C[C_1[N]], \\
 & & & & \delta = C_1[\underline{\gamma}], \\
 & & & & \gamma = N \\
 & & & & \downarrow \gamma \\
g_2 \equiv & \alpha = C[\underline{\delta}], & \xrightarrow[\delta]{} & g_3 \equiv & \alpha = C[C_1[N]], \\
 & \delta = C_1[N], & & & \delta = C_1[N], \\
 & \gamma = N & & & \gamma = N \ .
\end{array}
$$

□

Lemma 7.3 $\twoheadrightarrow_{\underline{\mathsf{as}}}$ *is strongly normalizing.*

Proof: Due to the fact that the ordering $>$ among recursion variables is well founded we can use the multiset ordering [Klo]. The weight associated to a system of recursion equations g is the multiset of all underlined recursion variables. *E.g.*, to the system

$$\alpha = \mathsf{F}\underline{\delta}, \delta = \mathsf{G}\underline{\gamma}, \gamma = 0 \ ,$$

we associate the multiset

$$\{\{\underline{\delta}, \underline{\gamma}\}\} \ .$$

Let

$$g \equiv \langle \alpha \mid \gamma = C[\underline{\delta}], \delta = M, E \rangle \longrightarrow_{\underline{\mathsf{as}}} \langle \alpha \mid \gamma = C[M], \delta = M, E \rangle \equiv g_1 \ .$$

Without loss of generality, let M be $C_1[\underline{\epsilon}]$. Then, in the multiset associated to g_1, $\underline{\delta}$ will be substituted by $\underline{\epsilon}$. By definition, $\gamma > \underline{\delta}$ and $\delta > \underline{\epsilon}$ and so the multiset is getting smaller. □

Theorem 7.4 *Acyclic substitution is confluent.*

Proof: As in [Bar84, Klo] confluence follows from Lemmas 7.2 and 7.3 by applying the complete development method, which consists of defining a new reduction relation with the same transitive closure as $\longrightarrow_{\mathsf{as}}$ and prove that it satisfies the diamond property. □

Next, we prove that acyclic substitution combined with β-reduction is confluent. We thus extend $\underline{\text{As}}$ by allowing underlining of λ's that constitute the operator part of a redex. The

new β-rule becomes $(\underline{\lambda}x.M)N \longrightarrow_{\underline{\beta}} M[x := N]$. The combination of $\underline{\text{as}}$ and $\underline{\beta}$ is written as $\twoheadrightarrow_{\underline{\text{as}\beta}}$. The new calculus is called $\underline{\lambda\Phi}$.

We start by showing that $\twoheadrightarrow_{\underline{\text{as}\beta}}$ is strongly normalizing. The proof follows the same steps as in [Bar84]. We associate a positive integer to each variable (recursion variables and lambda bound) occurring in the right-hand side of an equation of a system g. The weight of g, written as $|g|$, is then the sum of the weights occurring in g. However, the initial weight associated to variables has to obey some conditions.

Definition 7.5 Let g be a system of recursion equations in $\underline{\lambda\Phi}$. g has *decreasing weight property (dwp)* if

(i) for every $\underline{\beta}$-redex $(\underline{\lambda}x.P)Q$ in g :

$$\forall x \in P,\ |x| > |Q|$$

(ii) for every $\underline{\text{as}}$-redex $\underline{\gamma}$, such that $\gamma = M$ is an equation in g:

$$|\underline{\gamma}|>|M| \quad .$$

For example,

$$\alpha = (\underline{\lambda}x.x^{20})(\mathsf{G}\underline{\gamma}^{7}\underline{\gamma}^{8}), \gamma = \delta^{5}\delta^{1}$$

has the *dwp*, while

$$\alpha = (\underline{\lambda}x.x^{8})(\mathsf{G}\underline{\gamma}^{7}\underline{\gamma}^{1}), \gamma = \delta^{5}\delta^{1}$$

does not.

Proposition 7.6 *For all systems of recursion equations g in $\underline{\lambda\Phi}$, there exists an initial weight assignment so that g has decreasing weight property.*

Proof: We start by finding the strongly connected components of the graph associated to g. We could see the dag so obtained as having as nodes the sequences of equations that define a cyclic plane. These distinct sequences of equations are then topologically ordered, obtaining a new system of equations g'. The equations corresponding to each cyclic plane are not re-ordered. For example, the system

$$\alpha = \mathsf{F}\gamma, \delta = \mathsf{G}\eta, \gamma = \mathsf{H}\delta, \eta = \mathsf{H}\delta$$

is re-ordered as

$$\alpha = \mathsf{F}\gamma, \gamma = \mathsf{H}\delta, \delta = \mathsf{G}\eta, \eta = \mathsf{H}\delta$$

or

$$\alpha = \mathsf{F}\gamma, \gamma = \mathsf{H}\delta, \eta = \mathsf{H}\delta, \delta = \mathsf{G}\eta \ .$$

In other words, the order of the equations for δ and η is immaterial. Now we enumerate all the variables occurring in the right-hand side of the equations, following the right to left order, and assign to the mth variable occurrence the weight 2^m. Since

$$2^m \ > \ 2^{m-1} + 2^{m-2} + \cdots 2 + 1 \tag{7.4}$$

M has the *dwp*. □

Proposition 7.7 *If $g \longrightarrow_{\underline{\mathsf{as}\beta}} g_1$, and g has dwp then*

$$|g| > |g_1| \quad .$$

Proof: Follows from 7.4 above. □

Proposition 7.8 *Let $g \longrightarrow_{\underline{\mathsf{as}\beta}} g_1$, then if g has dwp so does g_1.*

Proof: If g reduces to g_1 by performing a $\underline{\beta}$-redex then the proof that the first condition of the *dwp* holds is the same as in [Bar84]. To show that the second condition holds let us assume:

$$\begin{array}{ll} \delta = C_3[\underline{\alpha}], & \longrightarrow_{\underline{\beta}} \ \delta = C_3[\underline{\alpha}], \\ \alpha = C[(\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}]], & \quad\ \ \alpha = C[C_1[\underline{\gamma}][x := C_2[\underline{\eta}]]], \\ \gamma = M, & \quad\ \ \gamma = M, \\ \eta = N & \quad\ \ \eta = N \ . \end{array}$$

Since during $\underline{\beta}$-reduction the weights of the recursion variables are not disturbed, we still have $|\underline{\gamma}|>|M|$ and $|\underline{\eta}|>|N|$. Since the weight of the right-hand side of α decreases, we still have $|\underline{\alpha}|>|C[C_1[\underline{\gamma}][x := C_2[\underline{\eta}]]]|$.

Let us now assume that g reduces to g_1 by performing an underlined acyclic substitution step. Let g be:

$$\begin{array}{l} \delta = C_3[\underline{\alpha}], \\ \alpha = C[(\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}]], \\ \gamma = M, \\ \eta = N \ . \end{array}$$

If we substitute for either η or γ then the first condition is met because the weight of the argument of the $\underline{\beta}$-redex decreases and x cannot occur free in M. Moreover, $|\underline{\alpha}|$ is still greater than the weight of the right-hand side. Analogously, if we substitute for $\underline{\alpha}$ we still have that the weights of $\underline{\gamma}$ and $\underline{\eta}$ are greater than $|M|$ and $|N|$, respectively. □

Lemma 7.9 *$\twoheadrightarrow_{\underline{\mathsf{as}\beta}}$ is strongly normalizing.*

Proof: From Propositions 7.7 and 7.8. □

Lemma 7.10 *Acyclic substitution commutes with β.*

Proof: We show that $\twoheadrightarrow_{\underline{\mathsf{as}}}$ commutes with $\twoheadrightarrow_{\underline{\beta}}$. Since $\twoheadrightarrow_{\underline{\mathsf{as}\beta}}$ is strongly normalizing it is enough to show that $\longrightarrow_{\underline{\mathsf{as}}}$ commutes with $\longrightarrow_{\underline{\beta}}$. Let g be

$$\begin{array}{l} \delta = C_3[\underline{\alpha}], \\ \alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}], \\ \gamma = M, \\ \eta = N \ . \end{array}$$

By cases on where the substitution occurs:

-Substitution for $\underline{\alpha}$.

$$
\begin{array}{ccc}
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}], \\
\gamma = M, \\
\eta = N
\end{array}
& \xrightarrow{\;\;\underline{\beta}\;\;} &
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]], \\
\gamma = M, \\
\eta = N
\end{array} \\
\downarrow \mathsf{as} & & \downarrow \mathsf{as} \\
\begin{array}{l}
\delta = C_3[(\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}]], \\
\alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}], \\
\gamma = M, \\
\eta = N
\end{array}
& \dashrightarrow_{\underline{\beta}} \;\; \dashrightarrow_{\underline{\beta}} &
\begin{array}{l}
\delta = C_3[(C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]]], \\
\alpha = (C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]], \\
\gamma = M, \\
\eta = N
\end{array}
\end{array}
$$

-Substitution for $\underline{\gamma}$.

$$
\begin{array}{ccc}
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}], \\
\gamma = M, \\
\eta = N
\end{array}
& \xrightarrow{\underline{\beta}} &
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]], \\
\gamma = M, \\
\eta = N
\end{array} \\
\downarrow \mathsf{as} & & \downarrow \mathsf{as} \\
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.C_1[M])C_2[\underline{\eta}], \\
\gamma = M, \\
\eta = N
\end{array}
& \dashrightarrow_{\underline{\beta}} &
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (C_1[M])[x := C_2[\underline{\eta}]], \\
\gamma = M, \\
\eta = N
\end{array}
\end{array}
$$

-Substitution for $\underline{\eta}$.

$$
\begin{array}{ccc}
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[\underline{\eta}], \\
\gamma = M, \\
\eta = N
\end{array}
& \xrightarrow{\underline{\beta}} &
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]], \\
\gamma = M, \\
\eta = N
\end{array} \\
\downarrow \mathsf{as} & & \downarrow \mathsf{as} \\
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.C_1[\underline{\gamma}])C_2[N], \\
\gamma = M, \\
\eta = N
\end{array}
& \dashrightarrow_{\underline{\beta}} &
\begin{array}{l}
\delta = C_3[\underline{\alpha}], \\
\alpha = (C_1[\underline{\gamma}])[x := C_2[N]], \\
\gamma = M, \\
\eta = N
\end{array}
\end{array}
$$

□

Theorem 7.11 *$\lambda\Phi$ is confluent.*

Proof: From the previous lemma and Hindley-Rosen's Lemma. □

Let us call a β-redex *spine-cyclic* when its root and the λ-node lie on the same cycle (see Figure 16). (The root of the β-redex and the λ-node may be cyclic.) Otherwise, the redex is *spine-acyclic*. Then, the above theorem says that reduction of spine-acyclic redexes is confluent, since they involve acyclic substitution only. An example of a spine-acyclic redex is the topmost redex of Figure 2. Confluence of $\lambda\Phi$ guarantees also that the lack of confluence of $\lambda\Theta$ does not impair its correctness, as shown next.

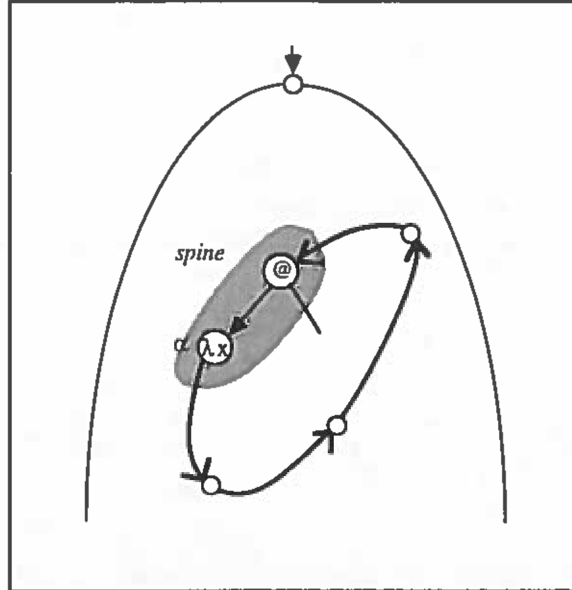


Figure 16: Cycle through a spine.

Theorem 7.12 *The λΘ-calculus is sound with respect to the infinitary lambda calculus.*

Proof: As shown in [AK95], if $g \longrightarrow_{\mathsf{c}} g_1$, $g \longrightarrow_{\mathsf{n}} g_1$ or $g \longrightarrow_{\mathsf{s}} g_1$, then g and g_1 are bisimilar graphs, and as such they have the same tree unwinding. We show next that β-reduction in $\lambda\Theta$ corresponds to a complete development of the residuals originating after unwinding a recursion system. Notation: $T^n(g)$ denotes the nth approximation of $T(g)$; $\longrightarrow_{\mathsf{cdv(as^*)}}$ denotes a complete development of all acyclic substitution redexes occuring in the first equation. $T^n(g)$ is obtained as follows: let g be $\alpha_1 = M_1, \cdots, \alpha_m = M_m$ we first create the new system $\alpha = \alpha_1, \alpha_1 = M_1, \cdots, \alpha_m = M_m$. If $\alpha = \alpha_1, \alpha_1 = M_1, \cdots, \alpha_m = M_m$ $\twoheadrightarrow_{\mathsf{cdv(as^*)}}$ $\alpha = M, \alpha_1 = M_1, \cdots, \alpha_m = M_m$ in n-steps, then the nth approximation of $T(g)$ is $M[\alpha_1 := \Omega, \cdots, \alpha_m := \Omega]$. The result then follows from the fact that acyclic substitution commutes with β and the descendant of an as^*-redex is still an as^*-redex:

$$g \longrightarrow_{\beta} g_1 \Longrightarrow \forall n, T^n(g) \twoheadrightarrow_{\beta} T^n(g_1) \ .$$

□

The other redex pointed at in Figure 2 is an example of a *spine-cyclic* redex, since the root of that redex and the λ-node, named **sum**, are on the same cyclic plane, and thus needing a substitution that is not acyclic to make it explicit. Implicit spine-cyclic redexes are made explicit by first applying the operation of copying, which allows to unwind a cycle without losing any name. For example, if we want to reduce the underlined implicit β-redex in the system:

$$\alpha = \lambda x.\underline{\delta(\mathsf{S}x)}, \delta = \lambda y.\alpha(\mathsf{S}y)$$

we first perform a copy step:

$$\alpha = \lambda x.\underline{\delta(\mathsf{S}x)}, \delta = \lambda y.\alpha(\mathsf{S}y) \longrightarrow_{\mathsf{c}} \alpha = \lambda x.\underline{\delta(\mathsf{S}x)}_1, \delta = \lambda y.\alpha'(\mathsf{S}y), \alpha' = \lambda x.\underline{\delta'(\mathsf{S}x)}_2, \delta' = \lambda y.\alpha'(\mathsf{S}y) \ .$$

The system so obtained now contains an implicit spine-acyclic redex, *i.e.*, the one subscripted with 1. However, another copy of the implicit spine-cyclic redex is made, *i.e.*, the one subscripted with 2.

Remark 7.13 Another notion of substitution that guarantees confluence is the *parallel substitution* ($\longrightarrow_{\mathsf{ps}}$), which consists of substituting at once for all the recursion variables.

$$\alpha_1 = M_1, \cdots, \alpha_n = M_n \longrightarrow_{\mathsf{ps}} \alpha_1 = M_1[\vec{\alpha_n}/\vec{M_n}], \cdots, \alpha_n = M_n[\vec{\alpha_n}/\vec{M_n}] \ .$$

For example, we have

$$\alpha = \lambda x.\delta(\mathsf{S}x), \delta = \lambda y.\alpha(\mathsf{S}y) \quad \longrightarrow_{\mathsf{ps}} \quad \alpha = \lambda x.(\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x), \delta = \lambda y.(\lambda x.\delta(\mathsf{S}x))(\mathsf{S}y) \ .$$

This notion is interesting since it allows to remove the definition of δ.

Since a notion of substitution is already present in the $\lambda\mu$-calculus, we are going to present it next.

7.1 The $\lambda\mu$-calculus

An interesting calculus arises by extending 'pure' λ-calculus with the μ-rule:

$$\mu: \ \mu x.Z(x) \longrightarrow Z(\mu x.Z(x)) \ .$$

Here we use the notation as used for 'higher-order term rewriting' by means of Combinatory Reduction Systems (CRSs), as in [KvOvR93]. Usually this rewriting rule is presented as $\mu x.Z \longrightarrow Z[x := \mu x.Z]$. The $\lambda\mu$-calculus already offers a form of cyclic λ-graph rewriting (see Figure 17); it reduces 'implicit' β-redexes in a way similar to that discussed in Section 4, that is, by first performing some unwinding. Thus, it seems puzzling that the $\lambda\mu$-calculus, being an orthogonal Combinatory Reduction System, is confluent. Translating the $\lambda\Theta$-counterexample

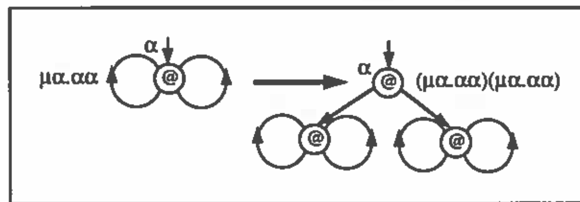


Figure 17: Reduction of $\mu\alpha.\alpha\alpha$.

to confluence of Section 4 into $\lambda\mu$ is instructive. The uppermost cyclic graph of Figure 9 is expressed in the $\lambda\mu$-calculus as:

$$M \equiv \mu\alpha.\lambda x.(\mu\delta.\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x) \ .$$

In order to reduce the (implicit) β-redex $\alpha(\mathsf{S}y)$, as we did in Section 4, we apply the μ-rule twice obtaining:

$$M \longrightarrow_\mu \lambda x.(\mu\delta.\lambda y.M(\mathsf{S}y))(\mathsf{S}x) \longrightarrow_\mu \lambda x.(\mu\delta.\lambda y.(\lambda x.(\mu\delta.\lambda y.M(\mathsf{S}y))(\mathsf{S}x))(\mathsf{S}y))(\mathsf{S}x) \ .$$

The above reduction is displayed in Figure 18. Note that the substitution operation embodied in the μ-rule is much more complex than the unrestricted version of $\lambda\Theta$ (displayed in Figure 19) and not as restrictive as acyclic substitution. The first μ-step has caused a copy of M. Furthermore, instead of simply substituting for α, another copy of M is made. This avoids the 'out-of-synch' phenomenon.

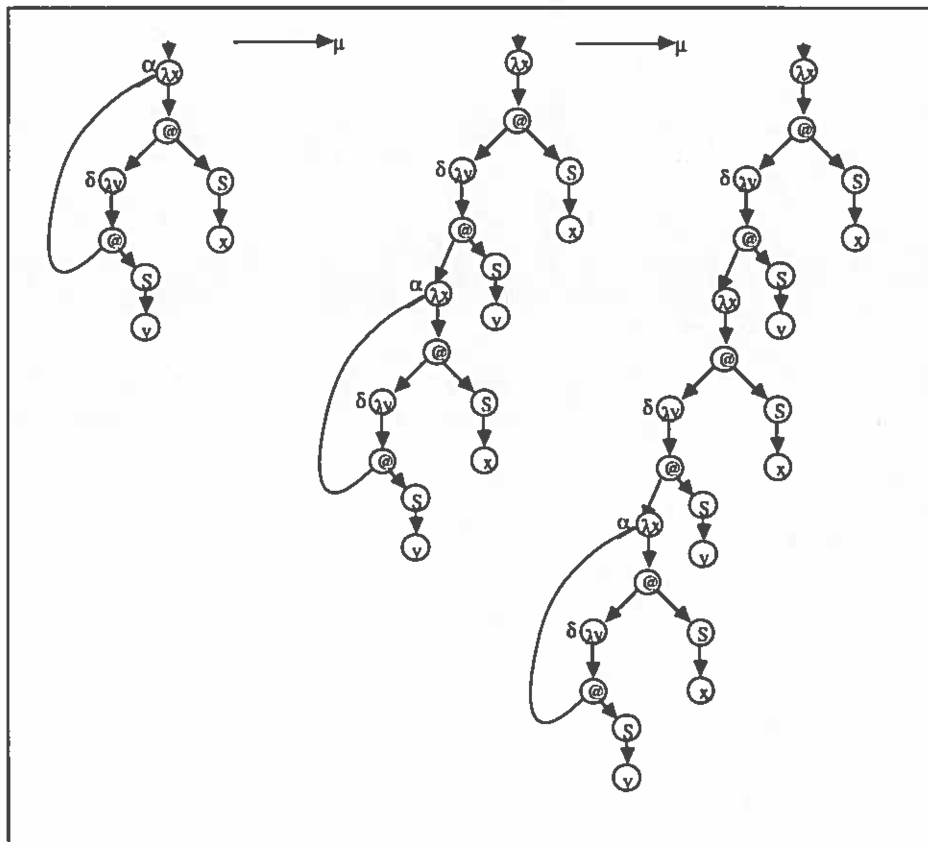


Figure 18: μ-reduction of $\mu\alpha.\lambda x.(\mu\delta.\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x)$.

At this point we could restrict ourselves to the sub-calculus $\lambda\mu$, however, this is not satisfactory because the $\lambda\mu$-calculus is limited in the form of sharing it can express. For example, it is unable to directly capture the expression

$$\alpha = \mathsf{F}(\gamma,\gamma),\ \gamma = \lambda x.\mathsf{G}(\gamma) \ .$$

In fact, by translating the above expression into the $\lambda\mu$-calculus we obtain

$$\mathsf{F}(\mu\gamma.\lambda x.\mathsf{G}(\gamma), \mu\gamma.\lambda x.\mathsf{G}(\gamma)) \ ,$$

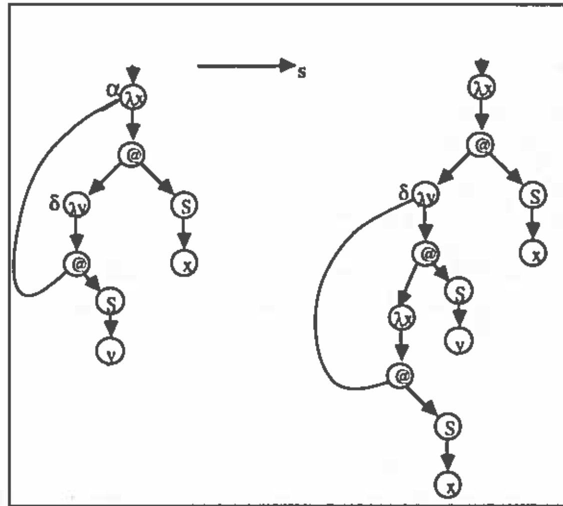


Figure 19: Display of a substitution step.

where a duplication or unsharing has occurred. In other words, the $\lambda\mu$-calculus expresses *vertical sharing* only. This gives rise to the following question: how can we extend $\lambda\mu$-calculus, with its lack of horizontal sharing, to include this feature that is indispensable for efficient graph rewriting, while retaining confluence and still properly extending well-known term rewriting techniques. This leads to modular lambda graph rewriting, which is introduced next.

8. Modular lambda graph rewriting

We now in a sequence of extensions develop a series of calculi, called $\lambda\phi$, leading to a very general and flexible calculus which incorporates the λ-calculus, the $\lambda\mu$-calculus, ordinary first-order term rewriting, vertical and horizontal sharing. The distinctive feature of this family of calculi is the presence of *nested recursion equations*. For example, we will write:

$$\langle \alpha \mid \alpha = (\lambda x.\langle \delta \mid \delta = \mathsf{F}(\alpha, \mathsf{S}x)\rangle)\mathsf{S}\underline{x}\rangle \ ,$$

where, as in Section 1, it is clear that the underlined x is free. To avoid free variable captures we will still assume that both free and bound variables have to be distinct from each other. So we will write the above term as

$$\langle \alpha \mid \alpha = (\lambda y.\langle \delta \mid \delta = \mathsf{F}(\alpha, \mathsf{S}y)\rangle)\mathsf{S}x\rangle \ .$$

Moreover, the root of a term is not restricted to be a variable, *e.g.*,

$$\langle \mathsf{F}\alpha \mid \alpha = \mathsf{G}0\rangle \ .$$

The general form of $\lambda\phi$-terms is

$$\langle t \mid E\rangle \ ,$$

where t is a term and E is an unordered sequence of equations; ϵ stands for the empty sequence. We refer to $\langle t \mid E\rangle$ as a box construct. We call t the *external part* of the box, and

E the *internal part.* We can see E as the environment associated to t, or as a set of delayed substitutions. The $\lambda\phi$-calculi can be seen as an extension of the $\lambda\sigma$-calculi [ACCL91, Cur93]; the $\lambda\sigma$-calculi treat the let-construct as a first class citizen, while the $\lambda\phi$-calculi support the letrec. For example, in $\lambda\phi$ we can have

$$\langle \alpha \mid \alpha = \lambda\gamma.\mathsf{F}(\delta,\gamma), \delta = \lambda\eta.\mathsf{G}(\alpha,\eta)\rangle,$$

which corresponds to the letrec expression:

$$\mathsf{letrec}\ \alpha = \lambda\gamma.\mathsf{F}(\delta,\gamma),\ \delta = \lambda\eta.\mathsf{G}(\alpha,\eta)\ \mathsf{in}\ \alpha\ \mathsf{end}\ .$$

We could also say that the $\lambda\sigma$-calculi express acyclic lambda graph rewriting, while the $\lambda\phi$-calculi deal with cyclic lambda graph rewriting. Since cycles are ubiquitous in the implementation of programming languages, the $\lambda\phi$-calculi follow the tradition of providing 'enriched λ-calculi' to capture more precisely the operational semantics of functional languages [Ari92, PJ87].

Definition 8.1 Let Σ be a first-order signature. Then $Ter_\Sigma(\lambda\phi)$, the set of $\lambda\phi$-terms over Σ, is given by:

(i) $\alpha, \gamma, \cdots\cdots \in Ter\Sigma(\lambda\phi)$;
(ii) $t_1, \cdots, t_n \in Ter_\Sigma(\lambda\phi) \Longrightarrow \mathsf{F}^n(t_1, \cdots, t_n) \in Ter_\Sigma(\lambda\phi)$;
(iii) $t \in Ter_\Sigma(\lambda\phi) \Longrightarrow \lambda\alpha.t \in Ter_\Sigma(\lambda\phi)$;
(iv) $t_0, t_1 \in Ter_\Sigma(\lambda\phi) \Longrightarrow t_0 t_1 \in Ter_\Sigma(\lambda\phi)$;
(v) $t_0, t_1, \cdots, t_n \in Ter_\Sigma(\lambda\phi) \Longrightarrow \langle t_0 \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle \in Ter_\Sigma(\lambda\phi)$, with $\alpha_i \neq \alpha_j, 0 \leq i < j \leq n$.

Note that we have dropped the distinction between lambda bound variables and recursion variables. α-equivalent $\lambda\phi$-terms are identified.

8.1 Graphical representation of modular λ-graphs

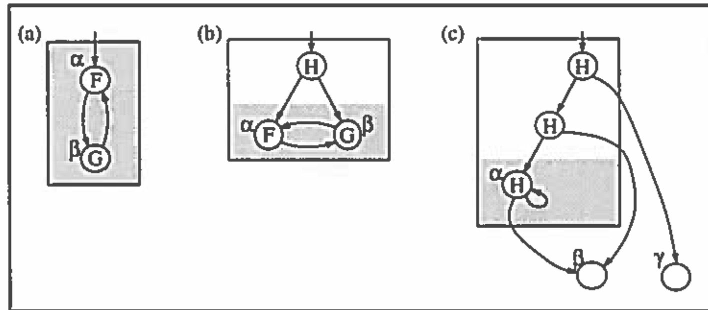


Figure 20: Graphs associated to $\lambda\phi$-terms.

We graphically represent an expression $\langle t \mid E\rangle$ by a box divided in two parts, the upper part corresponding to the external part t and the lower part containing the internal part E. A box can be thought of as a refined version of a node. We present a series of examples.

Example 8.2 (i) The following terms

$$(a)\ \langle \alpha \mid \alpha = \mathsf{F}(\beta), \beta = \mathsf{G}(\alpha) \rangle$$
$$(b)\ \langle \mathsf{H}(\alpha,\beta) \mid \alpha = \mathsf{F}(\beta), \beta = \mathsf{G}(\alpha) \rangle$$
$$(c)\ \langle \mathsf{H}(\mathsf{H}(\alpha,\beta),\gamma)) \mid \alpha = \mathsf{H}(\beta,\alpha) \rangle$$

are displayed in Figure 20. Note that the free variables are drawn outside the box, as in Figure 20(c).

(ii) The terms

$$(a)\ \langle \mathsf{H}(\langle \alpha \mid \alpha = \mathsf{F}(\alpha)\rangle, \beta) \mid \beta = \mathsf{F}(\langle \gamma \mid \gamma = \mathsf{F}(\delta), \delta = \mathsf{G}(\gamma)\rangle) \rangle$$
$$(b)\ \langle \mathsf{H}(\alpha',\beta) \mid \alpha' = \langle \alpha \mid \alpha = \mathsf{F}(\alpha)\rangle, \beta = \mathsf{F}(\gamma'), \gamma' = \langle \gamma \mid \gamma = \mathsf{F}(\delta), \delta = \mathsf{G}(\gamma)\rangle \rangle$$

are shown in Figure 21. Note the 'external names' α', γ' of the boxes in Figure 21(b).

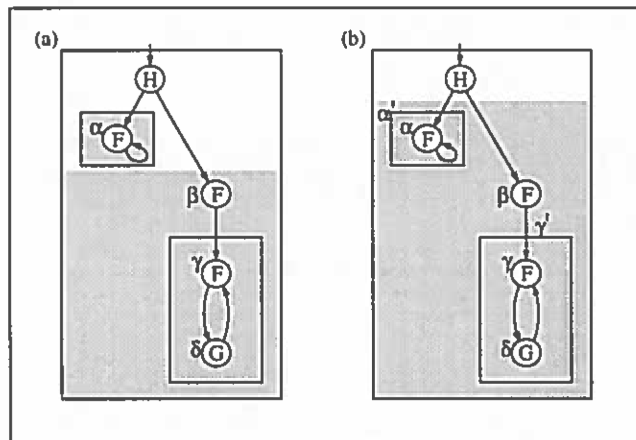


Figure 21: Graphs associated to $\lambda\phi$-terms.

(iii) Boxes can also refer to each other. The term

$$\langle \alpha' \mid \alpha' = \langle \alpha \mid \alpha = \mathsf{F}(\beta'), \beta' = \langle \beta \mid \beta = \mathsf{G}(\alpha')\rangle \rangle$$

is shown in Figure 22. Note that multiple references to a box are aiming straight at its leading node.

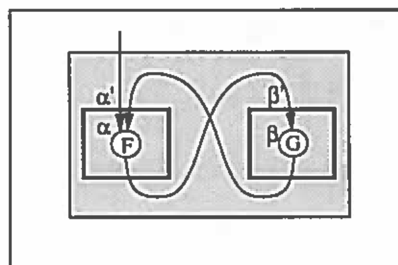


Figure 22: Mutually dependent cyclic boxes.

β-rule:			
$(\lambda\alpha.t)s$	$\longrightarrow_\beta$	$\langle t \mid \alpha = s\rangle$	
External substitution:			
$\langle C[\delta] \mid \delta = s, E\rangle$	$\longrightarrow_{\mathsf{es}}$	$\langle C[s] \mid \delta = s, E\rangle$	
Acyclic substitution:			
$\langle t \mid \alpha = C[\delta], \delta = s, E\rangle$	$\longrightarrow_{\mathsf{as}}$	$\langle t \mid \alpha = C[s], \delta = s, E\rangle$	if $\alpha > \delta$
Black hole:			
$\langle C[\delta] \mid \delta =_\circ \delta, E\rangle$	$\longrightarrow_\bullet$	$\langle C[\bullet] \mid \delta =_\circ \delta, E\rangle$	
$\langle t \mid \alpha = C[\delta], \delta =_\circ \delta, E\rangle$	$\longrightarrow_\bullet$	$\langle t \mid \alpha = C[\bullet], \delta =_\circ \delta, E\rangle$	if $\alpha > \delta$
Garbage collection rules:			
$t^{E,F}$	$\longrightarrow_{\mathsf{gc}}$	t^E	if $F \neq \epsilon$ and orthogonal to E and t
$\langle t \mid\,\rangle$	$\longrightarrow_{\mathsf{gc}}$	t	

Table 3: Reduction rules of $\lambda\phi_0$.

8.2 Basic System

We start with the basic system $\lambda\phi_0$. In order to simplify the reading of the reduction rules we will denote by t^E the term $\langle t \mid E\rangle$. Here "F orthogonal to a sequence of equations E and to a term t" means that the bound recursion variables of F do not intersect with the free variables of E and t. Contexts $C[\square]$ are given as follows:

$$C[\square] ::= \square \mid C[\square]t \mid \mathsf{F}(t_1, \cdots, C[\square], \cdots, t_n) \mid tC[\square] \mid \lambda\alpha.C[\square] \mid \langle C[\square] \mid E\rangle \mid \langle t \mid \alpha = C[\square], E\rangle\ .$$

The rewrite rules of $\lambda\phi_0$ are given in Table 3. In the β-rule notice the role change of the bound variable, previously bound by λ, afterwards bound by the recursion construct $\langle\ \mid\ \rangle$. The β-rule now becomes strongly normalizing. For example, $(\lambda\alpha.\alpha\alpha)(\lambda\alpha.\alpha\alpha) \longrightarrow_\beta \langle\alpha\alpha \mid \alpha = \lambda\alpha.\alpha\alpha\rangle$, which does not contain any β-redex. In order to proceed with the computation external substitution has to be applied, yielding: $\langle(\lambda\alpha.\alpha\alpha)(\lambda\alpha.\alpha\alpha) \mid \alpha = \lambda\alpha.\alpha\alpha\rangle$. *External substitution* allows us to 'extract' a tree-like prefix without duplicating the environment E. As for $\lambda\Theta$, the proviso $\alpha > \delta$ of the *Acyclic substitution* and *Black hole* rule indicates that there is no cycle between them in the term matching the left-hand side of the rule. For example, α is greater than δ in the term indicated with a curly brace:

$$g = \langle\gamma \mid \gamma = \underbrace{\langle\mathsf{F}\alpha \mid \alpha = \mathsf{G}\delta, \delta = \mathsf{G}\gamma\rangle}\rangle\ ,$$

even though g contains a cycle between α and δ. However, this cycle goes through γ, which is defined outside the internal box (see Figure 23). The binding $\delta =_\circ \delta$ in the *Black hole* rule stands for the sequence of bindings $\delta = \delta_1, \cdots, \delta_n = \delta$. The proviso "$F \neq \epsilon$" of the first *Garbage collection* rule guarantees its strong normalization. Without that proviso we would have $t^E \longrightarrow_{\mathsf{gc}} t^E$. Note that internal substitution as allowed in $\lambda\Theta$ is now absent. The system

$$\langle\alpha \mid \alpha = \lambda x.\gamma(\mathsf{S}x), \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle$$

does not rewrite in $\lambda\phi_0$ to

$$\langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}^2 x).\alpha(\mathsf{S}y)\rangle \ .$$

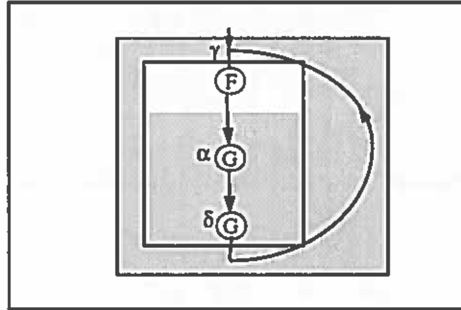
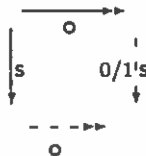


Figure 23: Ordering among recursion variables.

Theorem 8.3 *$\lambda\phi_0$ is confluent.*

Proof: Call the external and acyclic substitution rules s-reductions, and the remaining rules o-reductions. o-reductions are confluent, as they do not cause any duplication and they commute. As in Section 7, developments with respect to s-reductions are strongly normalizing and weak confluent. From Newman's Lemma, developments are thus confluent. Confluence of s-reductions follows from the complete development method. Next, one s-step commutes with a sequence of o-steps (Notation: $\longrightarrow^{0/1}$ stands for a reduction of 0 or 1 steps.):



Then we have that s-reductions commute with o-reductions. The result thus follows from Hindley-Rosen's Lemma. □

Lemma 8.4 *The λ-calculus is directly definable in $\lambda\phi_0$.*

Proof: We have:

$$(\lambda\alpha.t)s \longrightarrow_\beta \langle t \mid \alpha = s\rangle \twoheadrightarrow_{\mathsf{es}} \langle t[\alpha := s] \mid \alpha = s\rangle \twoheadrightarrow_{\mathsf{gc}} t[\alpha := s] \ .$$

The last step is justified by the fact that α cannot occur free in N. □

Theorem 8.5 *Let R be an orthogonal term rewriting system. Then $\lambda\phi_0 \cup R$ is confluent.*

Proof: Since R-rewriting commutes with $\lambda\phi_0$. □

As the following example suggests, rewriting with $\lambda\phi_0 \cup R$ is already quite interesting from the point of view of term graph rewriting, as it can handle both horizontal and vertical sharing.

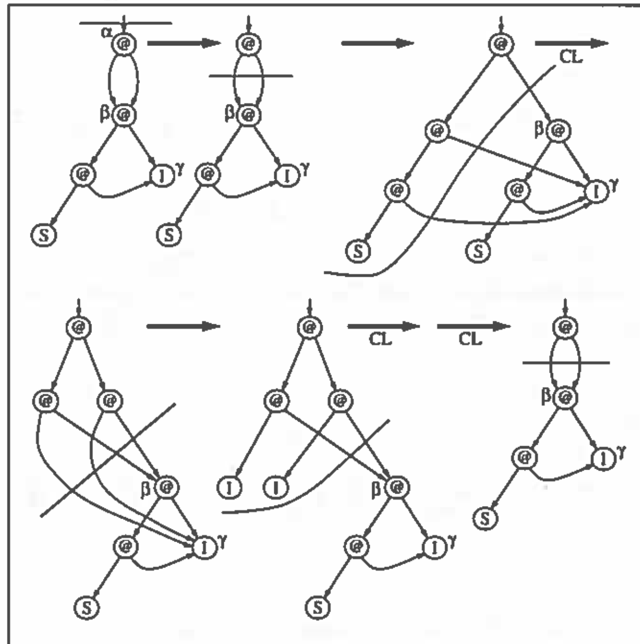


Figure 24: Reduction in $\lambda\phi_0 \cup R$.

Example 8.6 Consider CL, Combinatory Logic. Then we have the following reduction in $\lambda\phi_0 \cup R$ (see also Figure 24, where the lines dividing the graphs correspond to the division in external and internal part):

$$\begin{array}{ll}
\langle \alpha \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle & \longrightarrow_{\mathsf{es}} \\
\langle \beta\beta \mid \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle & \longrightarrow_{\mathsf{es}} \\
\langle \mathsf{S}\gamma\gamma\beta \mid \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle & \longrightarrow_{\mathsf{CL}} \\
\langle \gamma\beta(\gamma\beta) \mid \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle & \twoheadrightarrow_{\mathsf{es}} \\
\langle \mathsf{I}\beta(\mathsf{I}\beta) \mid \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle & \longrightarrow_{\mathsf{CL}} \\
\langle \beta\beta \mid \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I}\rangle &
\end{array}$$

8.3 $\lambda\mu$ with horizontal sharing

We translate the $\lambda\mu$-terms into $\lambda\phi_0$ as follows:

$$\begin{array}{lcl}
\phi\,[\![\alpha]\!] & = & \alpha \\
\phi\,[\![\mathsf{F}(t_1,\cdots,t_n)]\!] & = & \mathsf{F}(\phi\,[\![t_1]\!],\cdots,\phi\,[\![t_n]\!]) \\
\phi\,[\![t_1 t_2]\!] & = & \phi\,[\![t_1]\!]\,\phi\,[\![t_2]\!] \\
\phi\,[\![\lambda\alpha.t]\!] & = & \lambda\alpha.\phi\,[\![t]\!] \\
\phi\,[\![\mu\alpha.t]\!] & = & \langle \alpha \mid \alpha = \phi\,[\![t]\!]\rangle\ .
\end{array}$$

Note however that the $\lambda\mu$-calculus is not directly definable in $\lambda\phi_0$. *E.g.,* :

$$t \equiv \mu\alpha.\mathsf{F}(\mu\gamma.\mathsf{G}(\alpha,\gamma)) \longrightarrow_\mu \mu\alpha.\mathsf{F}(\mathsf{G}(\alpha,\mu\gamma.\mathsf{G}(\alpha,\gamma))) \equiv s\ ,$$

$(\lambda\alpha.t)^E$	$\longrightarrow_{\mathsf{d}\lambda}$	$\lambda\alpha.t^E$	
$\mathsf{F}^n(t_1,\cdots,t_n)^E$	$\longrightarrow_{\mathsf{dF}}$	$\mathsf{F}^n(t_1^E,\cdots,t_n^E)$	if $n \geq 1$
$(ts)^E$	$\longrightarrow_{\mathsf{d@}}$	$t^E s^E$	
$\langle\alpha \mid F\rangle^E$	$\longrightarrow_{\mathsf{d}\square}$	$\langle\alpha \mid F^E\rangle$	if α occurs bound in F

Table 4: Distribution rules.

but

$$\phi[\![t]\!] \equiv \langle\alpha \mid \alpha = \mathsf{F}(\langle\gamma \mid \gamma = \mathsf{G}(\alpha,\gamma)\rangle)\rangle \not\twoheadrightarrow_{\lambda\phi_0} \langle\alpha \mid \alpha = \mathsf{F}(\mathsf{G}(\alpha,\langle\gamma \mid \gamma = \mathsf{G}(\alpha,\gamma)\rangle))\rangle \equiv \phi[\![s]\!] \ .$$

To that end, we extend $\lambda\phi_0$ with the distribution rules of Table 4, whose job is to move a box construct as far as possible down a term until a variable is reached. We call the result $\lambda\phi_1$. Notation: F^E means: if F is $\alpha_1 = t_1,\cdots,\alpha_n = t_n$ then F^E is $\alpha_1 = t_1^E,\cdots,\alpha_n = t_n^E$.

Example 8.7 The following reduction

$$\mu\alpha.\mathsf{F}(\alpha,\mu\beta.\mathsf{G}(\alpha,\beta)) \longrightarrow \mu\alpha.\mathsf{F}(\alpha,\mathsf{G}(\alpha,\mu\beta.\mathsf{G}(\alpha,\beta)))$$

is defined in $\lambda\phi_1$ as follows:

$$\begin{array}{ll} \langle\alpha \mid \alpha = \mathsf{F}(\alpha,\langle\beta \mid \beta = \mathsf{G}(\alpha,\beta)\rangle)\rangle & \longrightarrow_{\mathsf{es}} \\ \langle\alpha \mid \alpha = \mathsf{F}(\alpha,\langle\mathsf{G}(\alpha,\beta) \mid \beta = \mathsf{G}(\alpha,\beta)\rangle)\rangle & \longrightarrow_{\mathsf{dF}} \\ \langle\alpha \mid \alpha = \mathsf{F}(\alpha,\mathsf{G}(\langle\alpha \mid \beta = \mathsf{G}(\alpha,\beta)\rangle,\langle\beta \mid \beta = \mathsf{G}(\alpha,\beta)\rangle))\rangle & \twoheadrightarrow_{\mathsf{gc}} \\ \langle\alpha \mid \alpha = \mathsf{F}(\alpha,\mathsf{G}(\alpha,\langle\beta \mid \beta = \mathsf{G}(\alpha,\beta)\rangle))\rangle \ . & \end{array}$$

(See Figure 25.)

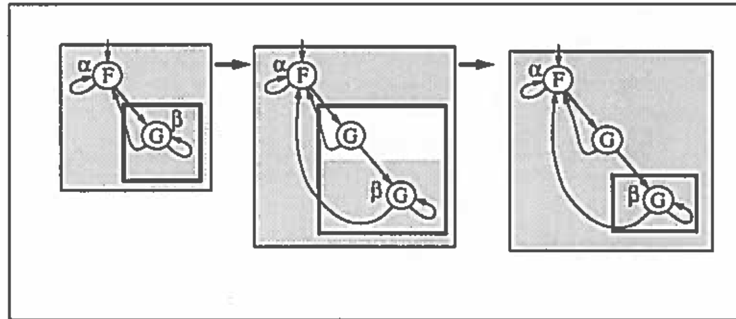


Figure 25: Analysis of μ-step.

In order to prove that $\lambda\mu$ is definable in $\lambda\phi_1$ we need some properties of the distribution and garbage collection rules. Specifically, that the distribution and garbage collection rules unfold the system by pushing the box constructs next to the variables. Notation: $\twoheadrightarrow_{\mathsf{dgc}}$ is the reduction relation induced by the distribution and garbage collection rules. We show next that $\twoheadrightarrow_{\mathsf{dgc}}$ is strongly normalizing and confluent.

Lemma 8.8 $\twoheadrightarrow_{\mathsf{dgc}}$ *is strongly normalizing.*

Proof: We associate to each box construct $\langle t \mid E\rangle$ a positive number n, called the index of $\langle t \mid E\rangle$. This index, written as $d(t)$, indicates the depth of the external part t of a box, that is, how much a box has to travel until it reaches a variable.

$$\begin{array}{lcl} d(\alpha) & = & 0 \\ d(constant) & = & 0 \\ d(st) & = & 1 + max\{d(s), d(t)\} \\ d(\mathsf{F}^n(t_1, \cdots, t_n)) & = & 1 + max\{d(t_1), \cdots, d(t_n)\}, n \geq 1 \\ d(\lambda\alpha.t) & = & 1 + d(t) \\ d(\langle t \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle) & = & 1 + d(t) + max\{d(t_1), \cdots, d(t_n)\} \ . \end{array}$$

(We assume $max\ \emptyset$ to be 0.) The index of each box appears as a superscript in the system below:

$$g \equiv \langle\langle\lambda\rho.\alpha\beta \mid \alpha = \langle\mathsf{F}\delta \mid \delta = \mathsf{G}\eta\rangle^1\rangle^2 \mid \eta = \langle\mathsf{G}\psi \mid \psi = 0\rangle^1\rangle^6 \ .$$

The weight associated with a system of recursion equations g, written as $|g|$, is then the multiset of sequences of indexes associated with all possible nesting of boxes. For example:

$$|g| = \{\!\{6\ 2\ 1, 6\ 1\}\!\} \ .$$

The multiset ordering is then induced by the lexicographic order on sequences. If a system of recursion equations g does not contain any box construct we let $|g|$ be $\{\!\{0\}\!\}$. The multiset ordering takes care of the duplication of boxes, *e.g.*, :

(i) If

$$g \equiv \langle\mathsf{H}(\alpha, \alpha) \mid \alpha = \langle\mathsf{FF}\beta \mid \beta = 1\rangle^2\rangle^1 \longrightarrow_{\mathsf{dF}}$$
$$\mathsf{H}(\langle\alpha \mid \alpha = \langle\mathsf{FF}\beta \mid \beta = 1\rangle^2\rangle^0, \langle\alpha \mid \alpha = \langle\mathsf{FF}\beta \mid \beta = 1\rangle^2\rangle^0) \equiv g_1 \ ,$$

then

$$|g| = \{\!\{1\ 2\}\!\} > |g_1| = \{\!\{0\ 2, 0\ 2\}\!\} \ ;$$

(ii) If

$$g \equiv \langle\langle\alpha \mid \alpha = \mathsf{H}(\mathsf{F}\beta, \mathsf{F}\delta), \beta = \mathsf{H}(\mathsf{F}\alpha, \mathsf{F}\delta)\rangle^0 \mid \delta = \langle\mathsf{F}\epsilon \mid \epsilon = 1\rangle^1\rangle^3 \longrightarrow_{\mathsf{d}\square}$$
$$\langle\alpha \mid \alpha = \langle\mathsf{H}(\mathsf{F}\beta, \mathsf{F}\delta) \mid \delta = \langle\mathsf{F}\epsilon \mid \epsilon = 1\rangle^1\rangle^2, \beta = \langle\mathsf{H}(\mathsf{F}\alpha, \mathsf{F}\delta) \mid \delta = \langle\mathsf{F}\epsilon \mid \epsilon = 1\rangle^1\rangle^2\rangle^0 \equiv g_1 \ ,$$

then

$$|g| = \{\!\{3\ 1, 3\ 0\}\!\} > |g_1| = \{\!\{0\ 2\ 1, 0\ 2\ 1\}\!\} \ .$$

We first restrict our attention to the distribution rules only. We show the following fact:

Fact: $C[R] \longrightarrow_{\mathsf{d}} C[R_1] \Longrightarrow d(C[R]) = d(C[R_1])$.

By induction on the structure of $C[\square]$.

-$C[\square] = \square$. By cases on R. Notation: if E is the sequence of equations $\alpha_1 = t_1, \cdots, \alpha_n = t_n$ then $d(E)$ stands for $max\{d(t_1), \cdots, d(t_n)\}$.

$$
\begin{aligned}
- d((\lambda\alpha.t)^E) &= 1 + d(\lambda\alpha.t) + d(E) \\
&= 1 + (1 + d(t) + d(E)) \\
&= 1 + d(t^E) \\
&= d(\lambda\alpha.t^E) \ .
\end{aligned}
$$

$$
\begin{aligned}
- d((st)^E) &= 1 + d(st) + d(E) \\
&= 1 + 1 + max\{d(s), d(t)\} + d(E) \\
&= 1 + max\{1 + d(s), 1 + d(t)\} + d(E) \\
&= 1 + max\{1 + d(s) + d(E), 1 + d(t) + d(E)\} \\
&= 1 + max\{d(s^E), d(t^E)\} = d(s^E t^E) \ .
\end{aligned}
$$

$$
\begin{aligned}
- d(\mathsf{F}^n(t_1, \cdots, t_n)^E) &= 1 + d(\mathsf{F}^n(t_1, \cdots, t_n)) + d(E) \\
&= 1 + 1 + max\{d(t_1), \cdots, d(t_n)\} + d(E) \\
&= 1 + max\{1 + d(E) + d(t_1), \cdots, 1 + d(E) + d(t_n)\} \text{ since } n \geq 1 \\
&= 1 + max\{d(t_1^E), \cdots, d(t_n^E)\} \\
&= d(\mathsf{F}^n(t_1^E, \cdots, t_n^E)) \ .
\end{aligned}
$$

Note that it is important for n to be greater than one, otherwise the depth would decrease in the reduction $\langle 0 \mid \rangle \longrightarrow_{\mathsf{dF}} 0$.

$$
\begin{aligned}
- d(\langle\langle\alpha \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle \mid E\rangle) &= 1 + d(\langle\alpha \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle) + d(E) \\
&= 1 + 1 + max\{d(t_1), \cdots, d(t_n)\} + d(E) \\
&= 1 + max\{1 + d(t_1) + d(E), \cdots, 1 + d(t_n) + d(E)\} \quad \text{since } n \geq 1 \\
&= 1 + max\{d(t_1^E), \cdots, d(t_n^E)\} \\
&= d(\langle\alpha \mid \alpha_1 = t_1^E, \cdots, \alpha_n = t_n^E\rangle) \ .
\end{aligned}
$$

The proviso of the $\longrightarrow_{\mathsf{d}\Box}$ -rule guarantees that n is greater than one.

-Inductive case. If $C[\Box]$ is $C_1[\Box]t$ then

$$
\begin{aligned}
d(C_1[R]t) &= 1 + max\{d(C_1[R]), d(t)\} \quad \text{Induction hypothesis} \\
&= 1 + max\{d(C_1[R_1]), d(t)\} \\
&= d(C_1[R_1]t) \ .
\end{aligned}
$$

The same for the other forms of $C[\Box]$.

We are now ready to show that

$$
g \equiv C[R] \longrightarrow_{\mathsf{d}} C[R_1] \equiv g_1 \Longrightarrow |g| > |g_1| \ .
$$

The proof is by induction on $C[\Box]$.

-$C[\Box] = \Box$. By cases on the rule being applied.

-$\langle\lambda\alpha.t \mid E\rangle \longrightarrow_{\mathsf{d}\lambda} \lambda\alpha.\langle t \mid E\rangle$. The index of the outside box is $1 + d(t)$ and it is replaced by $d(t)$. Any other box contained in t and in E is left unchanged.

-$\langle st \mid E\rangle \longrightarrow_{\mathsf{d}@} \langle s \mid E\rangle\langle t \mid E\rangle$. The index of the outside box is $1 + max\{d(s), d(t)\}$ and it is replaced by $d(s)$ and $d(t)$, respectively. The index of any other box contained in t, s and E is left unchanged.

-$\langle\mathsf{F}(t_1, \cdots, t_n) \mid E\rangle \longrightarrow_{\mathsf{dF}} \mathsf{F}(\langle t_1 \mid E\rangle, \cdots, \langle t_n \mid E\rangle)$. Same as the case above.

-$\langle\langle\alpha \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle \mid E\rangle \longrightarrow_{\mathsf{d}\Box} \langle\alpha \mid \alpha_1 = \langle t_1 \mid E\rangle, \cdots, \alpha_n = \langle t_n \mid E\rangle\rangle$. The index of the outside box is $1 + max\{d(t_1), \cdots, d(t_n)\}$ and it is replaced by 0.

- Inductive case. The only interesting case is when $C[\Box]$ is $\langle C_1[\Box] \mid E\rangle$, then according to the previous fact the index of the outside box does not increase. In other words, an internal reduction does not increase the index of the outside box.

Since a system of recursion equations g contains a finite number of equations and boxes, the garbage collection rules can be easily shown to be strongly normalizing. Let us assume there is an infinite sequence over the union of the distribution and garbage collection rules. This sequence can only have finitely many distribution steps. If not, since the garbage collection rules do not increase the weight of g, it means that the infinite sequence corresponds to an infinite descending chain. This is not possible. Thus, it must be that we have an infinite number of consecutive garbage collection steps, which contradicts the strong normalization of the garbage collection rules. □

Remark 8.9 If we change the current distribution rule over a box construct to

$$\langle t \mid E\rangle^F \longrightarrow \langle t^F \mid E^F\rangle \ ,$$

then the distribution rules will no longer be strongly normalizing. *E.g.*, :

$\langle\langle \mathsf{F}(\alpha,\delta) \mid \alpha = 0\rangle \mid \delta = 1\rangle \longrightarrow_{\mathsf{d}} \langle\langle \mathsf{F}(\alpha,\delta) \mid \delta = 1\rangle \mid \alpha = \langle 0 \mid \delta = 1\rangle\rangle \longrightarrow_{\mathsf{d}}$
$\langle\langle \mathsf{F}(\alpha,\delta) \mid \alpha = \langle 0 \mid \delta = 1\rangle\rangle \mid \delta = \langle 1 \mid \alpha = \langle 0 \mid \delta = 1\rangle\rangle\rangle \twoheadrightarrow_{\mathsf{d}} \langle\langle \mathsf{F}(\alpha,\delta) \mid \alpha = 0\rangle \mid \delta = 1\rangle \cdots$.

Lemma 8.10 $\twoheadrightarrow_{\mathsf{dgc}}$ *is confluent.*

Proof: The distribution rules define an orthogonal system, and thus are confluent. The garbage collection rules are themselves confluent. Since distribution and garbage collection rules commute, the result follows from Hindley-Rosen's Lemma. □

Notation: $t[\alpha_1 := \langle\alpha_1 \mid E\rangle, \cdots, \alpha_n := \langle\alpha_n \mid E\rangle]$ denotes a simultaneous substitution. $\mathsf{nf}_{\mathsf{dgc}}(t)$ is the normal form with respect to the distribution and garbage collection rules.

Lemma 8.11 (Unfolding Lemma) *Let t be a term and E be $\alpha_1 = s_1, \cdots, \alpha_n = s_n$. Then*

$$\langle t \mid E\rangle \twoheadrightarrow_{\mathsf{dgc}} \mathsf{nf}_{\mathsf{dgc}}(t)[\alpha_1 := \langle\alpha_1 \mid E\rangle, \cdots, \alpha_n := \langle\alpha_n \mid E\rangle] \ .$$

Proof: Trivial if E is empty. Otherwise, without loss of generality let us assume $n = 1$. Since $\twoheadrightarrow_{\mathsf{dgc}}$ is strongly normalizing we can conduct the proof by noetherian induction.

-t is a normal form. By structural induction on t.

-t is a variable. For t equal to α_1 the result follows trivially. Otherwise, let t be γ:

$$\langle\gamma \mid \alpha_1 = s_1\rangle \longrightarrow_{\mathsf{gc}} \langle\gamma \mid \rangle \longrightarrow_{\mathsf{gc}} \gamma \equiv \gamma[\alpha_1 := \langle\alpha_1 \mid \alpha_1 = s_1\rangle] \ .$$

-t is t_1t_2. We have:

$$\begin{array}{lll}
\langle t_1t_2 \mid \alpha_1 = s_1\rangle & \longrightarrow_{\mathsf{d@}} & \\
\langle t_1 \mid \alpha_1 = s_1\rangle\langle t_2 \mid \alpha_1 = s_1\rangle & \twoheadrightarrow_{\mathsf{dgc}} & \text{Induction hypothesis} \\
t_1[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle]t_2[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle] & \equiv & \\
(t_1t_2)[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle] \ . & &
\end{array}$$

-t is $\mathsf{F}(t_1, \cdots, t_n)$. Same as the case above.

-t is $\lambda\alpha.t_1$. We have:

$$\begin{array}{lll}
\langle \lambda\alpha.t_1 \mid \alpha_1 = s_1\rangle & \longrightarrow_{\mathsf{d}\lambda} & \\
\lambda\alpha.\langle t_1 \mid \alpha_1 = s_1\rangle & \twoheadrightarrow_{\mathsf{dgc}} & \text{Induction hypothesis} \\
\lambda\alpha.(t_1[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle]) & \equiv & \\
(\lambda\alpha.t_1)[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle] \ . & &
\end{array}$$

-t is $\langle \alpha_2 \mid \alpha_2 = s_2\rangle$. We have:

$$\begin{array}{lll}
\langle\langle \alpha_2 \mid \alpha_2 = s_2\rangle \mid \alpha_1 = s_1\rangle & \longrightarrow_{\mathsf{d}\square} & \\
\langle \alpha_2 \mid \alpha_2 = \langle s_2 \mid \alpha_1 = s_1\rangle\rangle & \twoheadrightarrow_{\mathsf{dgc}} & \text{Induction hypothesis} \\
\langle \alpha_2 \mid \alpha_2 = s_2[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle]\rangle & \equiv & \\
\langle \alpha_2 \mid \alpha_2 = s_2\rangle[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1\rangle] \ . & &
\end{array}$$

-t is not a normal form. Then:

$$\langle t \mid E\rangle \longrightarrow_{\mathsf{dgc}} \langle t' \mid E\rangle \ .$$

By induction hypothesis:

$$\langle t' \mid E\rangle \twoheadrightarrow_{\mathsf{dgc}} \mathsf{nf}_{\mathsf{dgc}}(t')[\alpha_1 := \langle \alpha_1 \mid E\rangle, \cdots, \alpha_n := \langle \alpha_n \mid E\rangle] \ .$$

From confluence of $\twoheadrightarrow_{\mathsf{dgc}}$ follows that $\mathtt{nf}_{\mathsf{dgc}}(t') \equiv \mathtt{nf}_{\mathsf{dgc}}(t)$.

□

Theorem 8.12 *$\lambda\mu$ is directly definable in $\lambda\phi_1$.*

Proof: We show that

$$\phi\,[\![\mu\alpha.t]\!] \twoheadrightarrow_{\lambda\phi_1} \phi\,[\![t[\alpha := \mu\alpha.t]]\!] \ .$$

$$\begin{array}{lll}
\phi\,[\![\mu\alpha.t]\!] & = & \text{Definition of } \phi \\
\langle \alpha \mid \alpha = \phi\,[\![t]\!]\rangle & \longrightarrow_{\mathsf{es}} & \\
\langle \phi\,[\![t]\!] \mid \alpha = \phi\,[\![t]\!]\rangle & \twoheadrightarrow_{\mathsf{dgc}} & \text{By the Unfolding Lemma} \\
\mathsf{nf}_{\mathsf{dgc}}(\phi\,[\![t]\!])[\alpha := \langle \alpha \mid \alpha = \phi\,[\![t]\!]\rangle] & = & \text{Since } \mathsf{nf}_{\mathsf{dgc}}(\phi\,[\![t]\!]) = \phi\,[\![t]\!] \\
\phi\,[\![t]\!]\,[\alpha := \langle \alpha \mid \alpha = \phi\,[\![t]\!]\rangle] & = & \text{Structural induction on } t \\
\phi\,[\![t[\alpha := \mu\alpha.t]]\!] \ . & &
\end{array}$$

Same for the β-rule. □

Next we want to show confluence of $\lambda\phi_1$. To that respect, we first need two propositions. Notation: $=_{\mathsf{dgc}}$ denotes the convertibility relation induced by the distribution and garbage collection rules.

Proposition 8.13 *Let t be a term, and E, F sequences of equations. Then,*

$$\langle\langle t \mid E\rangle \mid F\rangle =_{\mathsf{dgc}} \langle\langle t \mid F\rangle \mid E^F\rangle \ .$$

Proof: By noetherian induction on t with respect to the ordering induced by $\twoheadrightarrow_{\mathsf{dgc}}$.

- t is a normal form. By structural induction on t.
  - t is a variable α.
    If α is bound in E:

    $$\langle\langle \alpha \mid E\rangle \mid F\rangle \twoheadrightarrow_{\mathsf{d\Box}} \langle \alpha \mid E^F\rangle =_{\mathsf{gc}} \langle\langle \alpha \mid F\rangle \mid E^F\rangle \ .$$

    Otherwise:

    $$\begin{array}{ll} \langle\langle \alpha \mid E\rangle \mid F\rangle & \twoheadrightarrow_{\mathsf{gc}} \\ \langle \alpha \mid F\rangle & =_{\mathsf{gc}} \\ \langle\langle \alpha \mid F\rangle \mid E^F\rangle \ . & \end{array}$$

  - t is $t_1 t_2$.

    $$\begin{array}{lll} \langle\langle t_1 t_2 \mid E\rangle \mid F\rangle & \twoheadrightarrow_{\mathsf{d@}} & \\ \langle\langle t_1 \mid E\rangle \mid F\rangle\langle\langle t_2 \mid E\rangle \mid F\rangle & =_{\mathsf{dgc}} & \text{Induction hypothesis} \\ \langle\langle t_1 \mid F\rangle \mid E^F\rangle\langle\langle t_2 \mid F\rangle \mid E^F\rangle & =_{\mathsf{d@}} & \\ \langle\langle t_1 t_2 \mid F\rangle \mid E^F\rangle \ . & & \end{array}$$

  - t is $\mathsf{F}^n(t_1, \cdots, t_n)$. Same as the case above.
  - t is $\langle \alpha \mid E\rangle$. Without loss of generality, let us assume E to be $\alpha = s$.

    $$\begin{array}{lll} \langle\langle\langle \alpha \mid \alpha = s\rangle \mid E\rangle \mid F\rangle & \twoheadrightarrow_{\mathsf{d\Box}} & \\ \langle \alpha \mid \alpha = \langle\langle s \mid E\rangle \mid F\rangle\rangle & =_{\mathsf{dgc}} & \text{Induction hypothesis} \\ \langle \alpha \mid \alpha = \langle\langle s \mid F\rangle \mid E^F\rangle & =_{\mathsf{d\Box}} & \\ \langle\langle\langle \alpha \mid \alpha = s\rangle \mid F\rangle \mid E^F\rangle \ . & & \end{array}$$

- t is not a normal form. Follows immediately from the induction hypothesis.

□

Proposition 8.14 *Let s be a term and E a sequence of equations. Then,*

$$\langle C[s] \mid E\rangle =_{\mathsf{dgc}} \langle C[\langle s \mid E\rangle] \mid E\rangle \ .$$

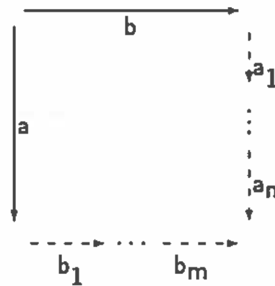
Proof: By structural induction on $C[\Box]$ and Proposition 8.13. □

Intermezzo 8.15 In the proof of confluence of $\lambda\phi_1$ we will use the decreasing diagram method proposed by Vincent [vO94]. The method consists of associating a label to each reduction step and giving a well-founded order on these labels. If all weakly confluent diagrams turn out to be of a specific kind, namely *decreasing*, then confluence is guaranteed.

Definition 8.16 Let $|.|$ be a measure from strings of labels to multisets of labels. If $a_1, \ldots, a_n$ are labels:

$$|a_1 \ldots a_n| = \{\!\{a_i | \text{ there is no } j < i \text{ with } a_j > a_i\}\!\} \ .$$

Then, the diagram



is decreasing if $\{\!\{a, b\}\!\} \geq |ab_1 \ldots b_m|$ and $\{\!\{a, b\}\!\} \geq |ba_1 \ldots a_n|$.

Theorem 8.17 *If a labelled reduction system is weakly confluent and all weakly confluent diagrams are decreasing with respect to a well founded order on labels then the system is confluent.*

Proof: See [vO94]. □

Theorem 8.18 *$\lambda\phi_1$ is confluent.*

Proof: We call the external and acyclic substitution s-reductions, and the remaining reductions, except β-reduction, o-reductions (written as $\twoheadrightarrow_{\mathsf{o}}$). Since the black-hole rule is strongly normalizing, and does not change the depth of a box, it follows that o-reductions are strongly normalizing. Their weak confluence thus implies confluence.

Let us study the new system, called $\lambda\phi'_1$, which contains the following rewrite rules:

$t \longrightarrow_{\mathsf{nf_o}} s$: if s is the normal form of t with respect to the o-rules;

$t \longrightarrow_{\|\mathsf{s}} s$: if s is obtained from t by a complete development of a set, possibly empty, of substitution redexes;

$t \longrightarrow_{\|\beta} s$: if s is obtained from t by a complete development of a set, possibly empty, of β redexes. Since β-reduction does not create new β-redexes, $t \longrightarrow_{\|\beta} s$ if and only if $t \twoheadrightarrow_{\beta} s$.

Let us first show the weak confluence diagrams.

-β-reduction and o-reductions. The goal is to show the following commuting diagram:

$\|\beta$ (top), $\mathsf{nf_o}$ (left), $\mathsf{nf_o}$ (right), $\|\beta$ $\mathsf{nf_o}$ (bottom) (8.5)

Let us first point out that the only obstacle to β commuting with o-reductions is caused by the distribution of an environment over an application:

$$((\lambda\alpha.s)t)^E \longrightarrow_{\mathsf{d@}} (\lambda\alpha.s)^E t^E \ .$$

The right-hand side of the reduction is no longer a β-redex. We call this distribution-step an interfering d@*-reduction. The distribution over lambda that restores the β-redex is denoted by dλ^*:

$$((\lambda\alpha.s)t)^E \longrightarrow_{\mathsf{d@*}} (\lambda\alpha.s)^E t^E \longrightarrow_{\mathsf{d}\lambda*} (\lambda\alpha.s^E)t^E \ .$$

If there is no interference, then a single o-reduction step commutes with β-reductions:

$$\begin{array}{ccc} \cdot & \xrightarrow{\ \ \|\beta\ \ } & \cdot \\ \downarrow \mathsf{o} & & \downarrow \mathsf{o} \\ \cdot & \xrightarrow[\ \ \|\beta\ \ ]{} & \cdot \end{array} \tag{8.6}$$

(Note that β-reduction does not cause any duplication.) Otherwise, we show the following:

$$\begin{array}{ccccc} \cdot & & \xrightarrow{\ \ \|\beta\ \ } & & \cdot \\ \downarrow \mathsf{d@*} & & & & \downarrow \mathsf{nf_o} \\ \cdot & \xrightarrow[\mathsf{d}\lambda*]{} \cdot & \xrightarrow[\|\beta]{} & \cdot \xrightarrow[\mathsf{nf_o}]{} & \cdot \end{array} \tag{8.7}$$

For a single β-step:

$$\begin{array}{ccccc} ((\lambda\alpha.t)s)^E & & \xrightarrow[\beta]{} & & \langle t \mid \alpha = s\rangle^E \\ \mathsf{d@*}\downarrow & & & & =_{\mathsf{dgc}} \\ (\lambda\alpha.t)^E s^E & \xrightarrow[\mathsf{d}\lambda*]{} & (\lambda\alpha.t^E)s^E & \xrightarrow[\beta]{} & \langle t^E \mid \alpha = s^E\rangle \end{array} \tag{8.8}$$

($\langle t \mid \alpha = s\rangle^E =_{\mathsf{dgc}} \langle t^E \mid \alpha = s^E\rangle$ follows from Proposition 8.13.) Since $\twoheadrightarrow_{\mathsf{dgc}}$ is confluent, $\langle t \mid \alpha = s\rangle^E$ and $\langle t^E \mid \alpha = s^E\rangle$ have the same normal form. For the number of β-steps greater than one we first re-order the β-reduction such that the interfering step is the last step. We then have:

$$\begin{array}{ccccc} \cdot & \xrightarrow[\beta]{\ n-1\ } & \cdot & \longrightarrow & \cdot \\ \downarrow d@ & (8.6) & \downarrow \mathsf{d@*} & (8.8) & \downarrow \mathsf{nf_o} \\ \cdot & \xrightarrow[\beta]{} & \cdot & & \\ \downarrow d\lambda & (8.6) & \downarrow d\lambda* & & \\ \cdot & \xrightarrow[\beta]{} & \cdot \xrightarrow[\beta]{} \cdot & \xrightarrow[\mathsf{nf_o}]{} & \cdot \end{array}$$

We are now ready to prove our result (*i.e.*, diagram (8.5)). Since o-reductions are strongly normalizing, the proof is by noetherian induction. The result holds trivially for a normal form. Otherwise, we have the following two cases:

$$\begin{array}{ccc} \cdot & \xrightarrow{\ \|\beta\ } & \cdot \\ {\scriptstyle o}\downarrow & (8.6) & \downarrow{\scriptstyle o} \\ \cdot & \xrightarrow{\ \|\beta\ } & \cdot \\ {\scriptstyle nf_o}\downarrow & I.H. & \downarrow{\scriptstyle nf_o} \\ \cdot & \xrightarrow{\|\beta}\ \cdot\ \xrightarrow{nf_o} & \cdot \end{array} \qquad \begin{array}{ccccc} \cdot & & \xrightarrow{\ \|\beta\ } & & \cdot \\ {\scriptstyle d@^*}\downarrow & & (8.7) & & \downarrow{\scriptstyle nf_o} \\ \cdot & \xrightarrow{d\lambda^*} & \cdot \xrightarrow{\ \|\beta\ } \cdot & \xrightarrow{nf_o} & \cdot \\ {\scriptstyle nf_o}\downarrow & & {\scriptstyle nf_o}\downarrow \quad I.H. \quad \downarrow{\scriptstyle nf_o} & & \\ \cdot & \equiv & \cdot \xrightarrow{\|\beta} \cdot \xrightarrow{nf_o} \cdot & & \end{array}$$

- **s-reductions and o-reductions.** The goal is to show the following commuting diagram:

$$\begin{array}{ccc} \cdot & \xrightarrow{\ \|s\ } & \cdot \\ {\scriptstyle nf_o}\downarrow & & \downarrow{\scriptstyle nf_o} \\ \cdot & \xrightarrow{\|s}\ \cdot\ \xrightarrow{nf_o} & \cdot \end{array} \tag{8.9}$$

We remind the reader that the bottom $\longrightarrow_{\|s}$ -reduction of the above diagram might correspond to an empty reduction. *E.g.*, :

$$\begin{array}{ccc} \langle t \mid \alpha = C[\delta], \delta = \delta_1, \delta_1 = \delta\rangle & \xrightarrow[as]{} & \langle t \mid \alpha = C[\delta_1], \delta = \delta_1, \delta_1 = \delta\rangle \\ \downarrow{\scriptstyle \bullet} & & \downarrow{\scriptstyle \bullet} \\ \langle t \mid \alpha = C[\bullet], \delta = \delta_1, \delta_1 = \delta\rangle & \equiv & \langle t \mid \alpha = C[\bullet], \delta = \delta_1, \delta_1 = \delta\rangle \ . \end{array}$$

The bottom $\longrightarrow_{nf_o}$ -step is due to the interference between external substitution and the distribution of an environment over a box construct:

$$\begin{array}{ccc} \langle\langle \alpha \mid \alpha = s\rangle \mid F\rangle & \xrightarrow[es]{\qquad} & \langle\langle s \mid \alpha = s\rangle \mid F\rangle \\ {\scriptstyle d\Box}\downarrow & & =_{dgc} \\ \langle \alpha \mid \alpha = \langle s \mid F\rangle\rangle & \xrightarrow[es]{} & \langle\langle s \mid F\rangle \mid \alpha = \langle s \mid F\rangle\rangle \ . \end{array}$$

The right-hand side of the top es-reduction is no longer a d□-redex. We call this external substitution an interfering es*-reduction. Analogously, we call this distribution over the box construct a d□*-reduction. A similar situation is caused by acyclic substitution:

$$\begin{array}{ccc} \langle\langle \alpha \mid \alpha = C[\delta], \delta = s\rangle \mid F\rangle & \xrightarrow[as]{\qquad} & \langle\langle \alpha \mid \alpha = C[s], \delta = s\rangle \mid F\rangle \\ {\scriptstyle d\Box}\downarrow & & \downarrow{\scriptstyle d\Box} \\ & & \langle\langle \alpha \mid \alpha = \langle C[s] \mid F\rangle, \delta = \langle s \mid F\rangle\rangle \\ & & =_{dgc} \\ \langle \alpha \mid \alpha = \langle C[\delta] \mid F\rangle, \delta = \langle s \mid F\rangle\rangle & \xrightarrow[as]{} & \langle \alpha \mid \alpha = \langle C[\langle s \mid F\rangle] \mid F\rangle, \delta = \langle s \mid F\rangle\rangle \ . \end{array}$$

Thus, the distribution of an environment E over a box construct of the form $\langle \alpha \mid F\rangle$ is interfering if $\langle \alpha \mid F\rangle$ is either an es or an as-redex.

By associating to each variable α a weight, say n, as in the proof of strong normalization of $\twoheadrightarrow_{\underline{\mathsf{as}\beta}}$, and to a variable α^n a depth of n instead of 0, we can show, following the steps of the proof of Lemma 8.8, that $\underline{\mathsf{s}}$-reductions (*i.e.*, developments with respect to the s-rules) combined with o-reductions are strongly normalizing. Then, by noetherian induction follows that, in case of non-interference, $\twoheadrightarrow_{\underline{\mathsf{s}}}$ commutes with $\twoheadrightarrow_{\mathsf{o}}$:

(8.10)

If the o-reduction interferes with $\underline{\mathsf{s}}$-reductions, we show the following:

where the $\underline{\mathsf{s}}$-reductions stand for complete developments. Let the d□*-redex be $\langle\langle \alpha \mid F\rangle \mid E\rangle$. We first re-order the $\underline{\mathsf{s}}$-reduction, such that, the external and acyclic substitution redexes that interfere with $\langle\langle \alpha \mid F\rangle \mid E\rangle$ are pushed at the end of the reduction. We then perform the descendants of the d□*-redex with respect to the non-interfering part of the $\underline{\mathsf{s}}$-reduction. We have:

Note that the n d□*-redexes are disjoint from each other, that is, the corresponding boxes are not contained into each other. We show by induction on n that we can close the above diagram.

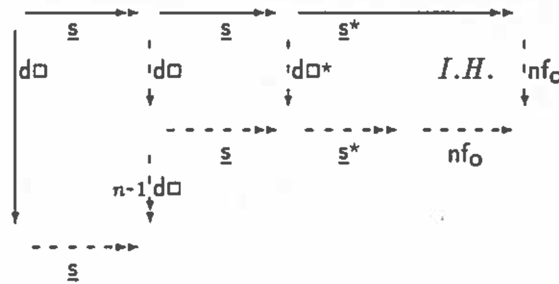
- $n = 1$: Without loss of generality, let F be $\alpha_1 = C_1[\underline{\alpha_2}], \alpha_2 = C_2[\underline{\alpha_3}], \alpha_3 = s$. We have:

$$\begin{array}{llcll}
\langle \underline{\alpha_1} \mid & \alpha_1 = C_1[\underline{\alpha_2}], & \xrightarrow{\underline{\mathsf{s}}^*} & \langle C_1[C_2[s]] \mid & \alpha_1 = C_1[C_2[s]], \\
& \alpha_2 = C_2[\underline{\alpha_3}], & & & \alpha_2 = C_2[s], \\
& \alpha_3 = s\rangle^E & & & \alpha_3 = s\rangle^E \\
& \downarrow \mathsf{d}\square^* & & & =_{\mathsf{dgc}} \\
\langle \underline{\alpha_1} \mid & \alpha_1 = C_1[\underline{\alpha_2}]^E, & \xrightarrow{\underline{\mathsf{s}}} & \langle C_1[C_2[s^E]^E]^E \mid & \alpha_1 = C_1[C_2[s^E]^E]^E, \\
& \alpha_2 = C_2[\underline{\alpha_3}]^E, & & & \alpha_2 = C_2[s^E]^E, \\
& \alpha_3 = s^E\rangle & & & \alpha_3 = s^E\rangle\ .
\end{array}$$

$$\langle C_1[C_2[s]] \mid \begin{array}{l}\alpha_1 = C_1[C_2[s]],\\ \alpha_2 = C_2[s],\\ \alpha_3 = s)^E\end{array} \;=_{\mathsf{dgc}}\; \langle C_1[C_2[s^E]^E]^E \mid \begin{array}{l}\alpha_1 = C_1[C_2[s^E]^E]^E,\\ \alpha_2 = C_2[s^E]^E,\\ \alpha_3 = s^E\rangle\end{array}$$

follows from Proposition 8.14.

- $n > 1$: We re-order the $\underline{\mathsf{s}}^*$-reduction, such that, the interfering steps with the first d□*-step are pushed at the end of the $\underline{\mathsf{s}}^*$-reduction. Note that this re-ordering does not cause a duplication of the d□*-redex. We thus have:



By re-ordering again the dashed middle $\underline{\mathsf{s}}$-reduction:

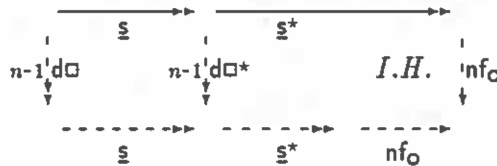
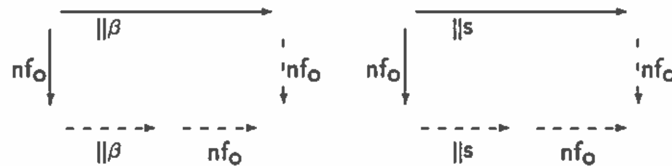
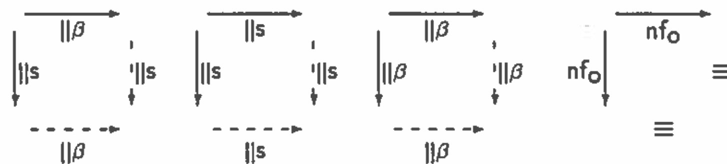


Diagram (8.9) follows by noetherian induction with respect to $\underline{\mathsf{s}}\cup\mathsf{o}$-reductions.

Summarizing, we have shown the following commuting diagrams:



Moreover, we also know the following ones:



According to the ordering $\| \beta > \mathsf{nf_o} < \| \mathsf{s}$, the above diagrams are decreasing, and thus by Theorem 8.17 $\lambda\phi_1'$ is confluent. Confluence of $\lambda\phi_1$ then follows from the following two points:

(1) Each rewrite rule of $\lambda\phi_1$ is a derived rule in $\lambda\phi_1'$. That is,

$$t \longrightarrow_{\lambda\phi_1} t_1 \Longrightarrow \exists s, t \twoheadrightarrow_{\lambda\phi_1'} s \text{ and } t_1 \twoheadrightarrow_{\lambda\phi_1'} s \;.$$

(2) Each reduction in $\lambda\phi'_1$ is contained in $\lambda\phi_1$:

$$t \twoheadrightarrow_{\lambda\phi'_1} t_1 \Longrightarrow t \twoheadrightarrow_{\lambda\phi_1} t_1 \ .$$

□

Intermezzo 8.19 $\lambda\phi_1$ extends the $\lambda\sigma$-calculus with names of Abadi et al. [ACCL91] with vertical sharing. We translate $\lambda\sigma$ into $\lambda\phi_1$ as follows:

$$\begin{array}{l}
\mathcal{T}[\![x]\!] = x \\
\mathcal{T}[\![ab]\!] = \mathcal{T}[\![a]\!]\,\mathcal{T}[\![b]\!] \\
\mathcal{T}[\![\lambda x.a]\!] = \lambda x.\mathcal{T}[\![a]\!] \\
\mathcal{T}[\![a[s]]\!] = \langle\langle \mathcal{T}[\![a]\!] \mid \mathcal{S}in_{\emptyset}(s)\rangle \mid \mathcal{S}out_{\emptyset}(s)\rangle
\end{array}$$

$$\begin{array}{l}
\mathcal{S}in_{var}[\![id]\!] = \epsilon \\
\mathcal{S}in_{var}[\![(a/x).s]\!] = \begin{cases} \mathcal{S}in_{var}(s) & x \in var \\ x = x', \mathcal{S}in_{var\cup\{x\}}(s) & x \notin var \end{cases} \\
\mathcal{S}out_{var}[\![id]\!] = \epsilon \\
\mathcal{S}out_{var}[\![(a/x).s]\!] = \begin{cases} \mathcal{S}out_{var}(s) & x \in var \\ x' = \mathcal{T}[\![a]\!], \mathcal{S}out_{var\cup\{x\}}(s) & x \notin var \ . \end{cases}
\end{array}$$

The above translation indicates how to map a let construct into a letrec. Namely, in order to avoid variable capture, each binding has to be split in two. For example, the term

$$\mathsf{let}\ x = \mathsf{cons}\ 1\ x\ \mathsf{in}\ x$$

is translated as

$$\mathsf{letrec}\ x' = \mathsf{cons}\ 1\ x\ \mathsf{in}\ \mathsf{letrec}\ x = x'\ \mathsf{in}\ x \ .$$

The binding $x = x'$ is generated by $\mathcal{S}in_{var}$ and the binding $x' = \mathsf{cons}\ 1\ x$ is generated by $\mathcal{S}out_{var}$.

The substitution rules and garbage collection rules of $\lambda\phi_1$ simulate the lookup of a variable in a substitution, which is expressed in $\lambda\sigma$ by the following rules:

$$\begin{array}{lcll}
\mathit{Var}_1: & & & \\
\quad x[(a/x).s] & = & a & \\
\mathit{Var}_2: & & & \\
\quad x[(a/y).s] & = & x[s] & \text{if } x \neq y \\
\mathit{Var}_3: & & & \\
\quad x[id] & = & x \ . &
\end{array}$$

Var_1 entails that the $\lambda\sigma$-calculus does not deal with cyclic substitutions. The distribution rules simulate the following rules:

$$\begin{array}{lcll}
\mathit{Abs}: & & & \\
\quad (\lambda x.a)[s] & = & \lambda y.a[(y/x).s] & \text{if } y \text{ occurs in neither } a \text{ nor } s \\
\mathit{App}: & & & \\
\quad (ab)[s] & = & (a[s])(b[s]) \ . &
\end{array}$$

8.4 A calculus for modular lambda graph rewriting

Yet, from the point of view of efficiency, $\lambda\phi_1$ is not quite satisfactory. Consider the following reduction:

$$\langle(\alpha 3)+(\alpha 4) \mid \alpha = \langle \lambda\gamma.\gamma * \delta \mid \delta = t\rangle\rangle \twoheadrightarrow \langle 3 * \delta \mid \delta = t\rangle + \langle 4 * \delta \mid \delta = t\rangle \ .$$

An unnecessary copy of t has been performed; no occurrence of the bound variable γ can occur in t and thus its computation can be shared. In fact, this is the essence of lazy and fully lazy interpreters [HM76, Wad71, Tur79, AKP84]. This leads to the desire of eliminating the internal boxes. To that respect, we need to distinguish between two kind of boxes, *acyclic* and *cyclic*. The boxes of Figure 25 that are drawn with heavy lines are examples of cyclic boxes. On the other hand, the boxes of Figure 26 are acyclic. The reason is that the cyclic path is required to go through the internal part of the box and to be within the *parent box*. (A parent box of a box is the smallest box properly containing it.) The distinction between cyclic and acyclic boxes is important since only acyclic boxes can be removed safely. The reason is that once a cyclic box is removed then operations on that cycle will no longer be allowed, as shown below:

$$\langle\gamma \mid \gamma = \langle \mathsf{F}\alpha \mid \alpha = \mathsf{G}\delta, \delta = \mathsf{G}\gamma\rangle\rangle \xrightarrow[\mathsf{asugc}]{} \langle\gamma \mid \gamma = \langle \mathsf{F}\alpha \mid \alpha = \mathsf{GG}\gamma\rangle\rangle \xrightarrow[\mathsf{esugc}]{} \langle\gamma \mid \gamma = \mathsf{FGG}\gamma\rangle$$

$$\downarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$\langle\gamma \mid \gamma = \mathsf{F}\alpha, \alpha = \mathsf{G}\delta, \delta = \mathsf{G}\gamma\rangle \dashrightarrow \ ?$$

We have removed the internal box of $\langle\gamma \mid \gamma = \langle \mathsf{F}\alpha \mid \alpha = \mathsf{G}\delta, \delta = \mathsf{G}\gamma\rangle\rangle$ which, as depicted in Figure 23, is on a cycle. Once this cyclic box is removed the substitutions for α and δ will no longer be acyclic substitutions. Note that boxes of the form $\langle t \mid\rangle$ can always be safely removed. What about the internal box in the following term

$$\langle\alpha \mid \alpha = \lambda\gamma.\langle\delta + \delta \mid \delta = \gamma * \gamma\rangle\rangle \ .$$

It cannot be removed since γ will get out of scope. In fact, we can see that internal box as a cyclic box by representing each reference to a bound variable as a link back to the corresponding λ-node, as in [Wad71].

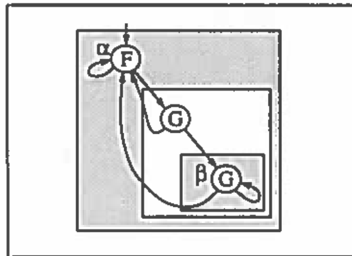


Figure 26: Graph of $\langle\alpha \mid \alpha = \mathsf{F}(\alpha, \langle \mathsf{G}(\alpha, \langle\beta \mid \beta = \mathsf{G}(\alpha,\beta)\rangle) \mid\rangle)\rangle$.

The box removal procedure at the graphical representation is simple: a box is physically erased in one act. At the textual level, let us first propose the following rules:

$$(t^E)^F \longrightarrow_{\Box\mathsf{m}} t^{E,F}$$

$$\langle t \mid \alpha = s^E, F\rangle \longrightarrow_{\Box\Box} \langle t \mid \alpha = s, E, F\rangle \ \text{ if } s^E \text{ is acyclic} \ .$$

$\lambda\alpha.t^{E_1,E}$	$\longrightarrow_{\Box\lambda}$	$(\lambda\alpha.t^{E_1})^E$	if $E \neq \epsilon$ and $E_1, \alpha > E$
$\mathsf{F}(t_1,\cdots,t_i^E,\cdots,t_n)$	$\longrightarrow_{\Box\mathsf{F}}$	$\mathsf{F}(t_1,\cdots,t_i,\cdots,t_n)^E$	
$t^E s$	$\longrightarrow_{\Box @_l}$	$(ts)^E$	
ts^E	$\longrightarrow_{\Box @_r}$	$(ts)^E$	
$(t^E)^F$	$\longrightarrow_{\Box m}$	$t^{E,F}$	
$\langle t \mid \alpha = s^{E_1,E}, F\rangle$	$\longrightarrow_{\Box\Box}$	$\langle t \mid \alpha = s^{E_1}, E, F\rangle$	if $E \neq \epsilon$ and $E_1, \alpha > E$

Table 5: Box elimination rules.

We then face a problem if in the example below s^E is cyclic:

$$\begin{array}{ccc} \langle t \mid \alpha = (s^E)^F\rangle & \xrightarrow[\Box m]{} & \langle t \mid \alpha = s^{E,F}\rangle \\ \downarrow \Box\Box & & \vdots \\ \langle t \mid \alpha = s^E, F\rangle & \dashrightarrow & ? \end{array}$$

Thus, in order for confluence not to fail we need to be able to move out from a box the equations that are not on a cycle, as shown in the rule:

$$\langle t \mid \alpha = s^{E_1,E}, F\rangle \longrightarrow_{\Box\Box} \langle t \mid \alpha = s^{E_1}, E, F\rangle \qquad \text{if } E \neq \epsilon \text{ and } E_1, \alpha > E \enspace ,$$

where $\alpha > E$ means that α and the bound recursion variables of E do not lie on the same cyclic plane; $E_1 > E$ means that the recursion variables of E_1 do not occur free in E. Equipped with this rule we can now close the above diagram:

$$\begin{array}{ccc} \langle t \mid \alpha = (s^E)^F\rangle & \xrightarrow[\Box m]{} & \langle t \mid \alpha = s^{E,F}\rangle \\ \downarrow \Box\Box & & \vdots\ \Box\Box \\ \langle t \mid \alpha = \langle s^E \mid \rangle, F\rangle & \underset{\mathsf{gc}}{\dashrightarrow} & \langle t \mid \alpha = s^E, F\rangle \end{array} \enspace .$$

The box elimination rules are displayed in Table 5. The proviso $E \neq \epsilon$ is to guarantee strong normalization. Since box elimination causes more sharing, it means that if we want confluence to hold we need to introduce an operation that unshares the system. We thus admit the operation of copying. The new system is called $\lambda\phi_2$. We give all the reduction rules of $\lambda\phi_2$ in Table 6. The horizontal line suggests that the rules below the line can be considered as part of a canonicalization procedure. Since $\lambda\phi_2$ turns out to be confluent, we can do rewriting on terms that do not contain any syntactic noise, such as the presence of garbage and acyclic boxes.

Proposition 8.20 *The box elimination rules, garbage collection and black hole are strongly normalizing.*

Proof: We first show that the box elimination rules are strongly normalizing. To each term t we associate the following measure:

$$|t| = w(t) \enspace ,$$

where $w(t)$ is the multiset of weights associated to each equation and box in t. $w(t)$ is defined as follows:

$$\begin{array}{lcl} w(\lambda\alpha.t) & = & inc(w(t)) \\ w(\mathsf{F}^n(t_1,\cdots,t_n)) & = & inc(w(t_1)\cup\cdots\cup w(t_n)) \\ w(st) & = & inc(w(s)\cup w(t)) \\ w(\alpha) & = & \emptyset \\ w(\langle t \mid \alpha_1 = t_1,\cdots,\alpha_n = t_n\rangle) & = & w(t)\cup \{\!\{\underbrace{0,\cdots,0}_{n+1}\}\!\} \cup inc(w(t_1),\cdots,w(t_n)) \ . \end{array}$$

inc adds one to each element of the multiset, *i.e.*, $inc(\{\!\{n_1,\cdots,n_m\}\!\}) = \{\!\{n_1+1,\cdots,n_m+1\}\!\}$. For example, $|\langle\langle\alpha \mid \alpha = \mathsf{F}\beta\rangle \mid \beta = \langle\delta \mid \delta = \mathsf{G}\delta\rangle\rangle| = \{\!\{0,0,0,0,1,1\}\!\}$. Is then routine to check that this measure decreases at each box elimination step, and does not increase with garbage collection and black hole. It thus follows that their union is strongly normalizing. □

Proposition 8.21 *The box elimination rules with garbage collection and black hole are confluent.*

Proof: Follows from the fact that all critical pairs converge and from strong normalization. □

Theorem 8.22 *$\lambda\phi_2$ is confluent.*

Proof: As in the proof of confluence of $\lambda\phi_1$, we first prove confluence of a new system, called $\lambda\phi_2'$, which contains the following rewrite rules:

$t \longrightarrow_{\mathsf{nf}_{\square\mathsf{e}}} s$: if s is the normal form of t with respect to the box elimination rules, black hole and garbage collection rules;

$t \longrightarrow_{\mathsf{nf}_{\mathsf{o}}} s$: if s is the normal form of t with respect to the distribution rules, black hole and garbage collection rules;

$t \longrightarrow_{\mathsf{cgc}} s$: if s is obtained from t by performing a copy step followed by the reduction to normal form with respect to garbage collection;

$t \longrightarrow_{\|\mathsf{s}} s$: if s is obtained from t by a complete development of a set, possibly empty, of substitution redexes;

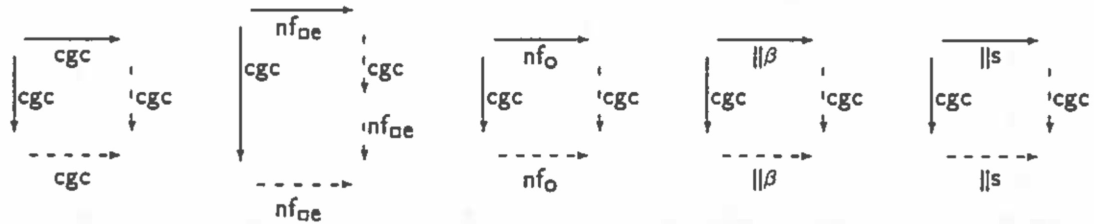
$t \longrightarrow_{\|\beta} s$: if s is obtained from t by a complete development of a set, possibly empty, of β redexes.

We show next the weak confluence diagrams.

β-rule:			
$(\lambda\alpha.t)s$	$\longrightarrow_{\beta}$	$\langle t \mid \alpha = s\rangle$	
External substitution:			
$\langle C[\delta] \mid \delta = s, E\rangle$	$\longrightarrow_{\mathsf{es}}$	$\langle C[s] \mid \delta = s, E\rangle$	
Acyclic substitution:			
$\langle t \mid \alpha = C[\delta], \delta = s, E\rangle$	$\longrightarrow_{\mathsf{as}}$	$\langle t \mid \alpha = C[s], \delta = s, E\rangle$	if $\alpha > \delta$
Distribution rules:			
$(\lambda\alpha.t)^E$	$\longrightarrow_{\mathsf{d}\lambda}$	$\lambda\alpha.t^E$	
$\mathsf{F}^n(t_1, \cdots, t_n)^E$	$\longrightarrow_{\mathsf{dF}}$	$\mathsf{F}^n(t_1^E, \cdots, t_n^E)$	if $n \geq 1$
$(ts)^E$	$\longrightarrow_{\mathsf{d@}}$	$t^E s^E$	
$\langle \alpha \mid F\rangle^E$	$\longrightarrow_{\mathsf{d}\Box}$	$\langle \alpha \mid F^E\rangle$	if α occurs bound in F
Copying			
t	$\longrightarrow_{\mathsf{c}}$	s	if $\exists$ a variable mapping $\sigma, s^\sigma = t$
Black hole:			
$\langle C[\delta] \mid \delta =_\circ \delta, E\rangle$	$\longrightarrow_{\bullet}$	$\langle C[\bullet] \mid \delta =_\circ \delta, E\rangle$	
$\langle t \mid \alpha = C[\delta], \delta =_\circ \delta, E\rangle$	$\longrightarrow_{\bullet}$	$\langle t \mid \alpha = C[\bullet], \delta =_\circ \delta, E\rangle$	if $\alpha > \delta$
Garbage collection rules:			
$t^{E,F}$	$\longrightarrow_{\mathsf{gc}}$	t^E	if $F \neq \epsilon$ and orthogonal to E and t
$\langle t \mid \rangle$	$\longrightarrow_{\mathsf{gc}}$	t	
Box elimination rules:			
$\lambda\alpha.t^{E_1,E}$	$\longrightarrow_{\Box\lambda}$	$(\lambda\alpha.t^{E_1})^E$	if $E \neq \epsilon$ and $E_1, \alpha > E$
$\mathsf{F}(t_1, \cdots, t_i^E, \cdots, t_n)$	$\longrightarrow_{\Box\mathsf{F}}$	$\mathsf{F}(t_1, \cdots, t_i, \cdots, t_n)^E$	
$t^E s$	$\longrightarrow_{\Box@_l}$	$(ts)^E$	
ts^E	$\longrightarrow_{\Box@_r}$	$(ts)^E$	
$(t^E)^F$	$\longrightarrow_{\Box\mathsf{m}}$	$t^{E,F}$	
$\langle t \mid \alpha = s^{E_1,E}, F\rangle$	$\longrightarrow_{\Box\Box}$	$\langle t \mid \alpha = s^{E_1}, E, F\rangle$	if $E \neq \epsilon$ and $E_1, \alpha > E$

Table 6: Reduction rules of $\lambda\phi_2$.

$\longrightarrow_{\mathsf{cgc}}$ **and the other rules:**



The confluence of copying is shown in [AK95]. Copy does not commute with $\longrightarrow_{\mathsf{nf}_{\Box\mathsf{e}}}$ because a copy step can turn some cyclic boxes into acyclic boxes, as shown next.

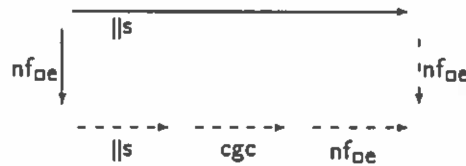
$$t \equiv \langle \alpha \mid \alpha = \langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha \rangle\rangle \longrightarrow_{\mathsf{c}} \langle \alpha \mid \alpha = \underbrace{\langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha' \rangle}, \alpha' = \langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha' \rangle\rangle \equiv s \ .$$

t is in normal form with respect to the box elimination rules. On the other hand, s contains one box, the one underbraced, that can be eliminated.

$\longrightarrow_{\mathsf{nf}_{\Box\mathsf{e}}}$ **and** $\longrightarrow_{\|\mathsf{s}}$ **.** The obstacle to $\longrightarrow_{\mathsf{nf}_{\Box\mathsf{e}}}$ commuting with $\longrightarrow_{\|\mathsf{s}}$ is due to the following interference (s' indicates a renamed version of s.):

$$\begin{array}{ccccc}
\langle C[\delta] \mid \delta = s^{E,F} \rangle & \xrightarrow[\mathsf{es}]{\hspace{6em}} & & & \langle C[s^{E,F}] \mid \delta = s^{E,F} \rangle \\
\Box_\Box \downarrow & & & & =_{\mathsf{nf}_{\Box\mathsf{e}}} \\
\langle C[\delta] \mid \delta = s^{E}, F \rangle & \dashrightarrow_{\mathsf{es}} & \langle C[s^{E}] \mid \delta = s^{E}, F \rangle & \dashrightarrow_{\mathsf{c}} & \langle C[s'^{E'}] \mid \delta = s^{E}, F, F' \rangle
\end{array}$$

The same happens in case of acyclic substitution. In other words, $\longrightarrow_{\Box_\Box}$ interferes with substitution if $s^{E,F}$ is involved in the substitution. Following a similar argument as in the study of the interaction between $\longrightarrow_{\mathsf{d}\Box}$ and the substitution rules, and from the interaction between cgc and $\mathsf{nf}_{\Box_\mathsf{e}}$ we have the following commuting diagram:



$\longrightarrow_{\mathsf{nf}_{\Box_\mathsf{e}}}$ **and** $\longrightarrow_{\mathsf{nf}_\circ}$ **.** Let us first analyze each distribution rule.

$\longrightarrow_{\mathsf{d}\lambda}$ **.**

$$\begin{array}{ccc}
(\lambda\alpha.t)^E & \xrightarrow[\mathsf{d}\lambda]{} & \lambda\alpha.t^E \\
\equiv & & \downarrow \Box_\lambda \\
(\lambda\alpha.t)^E & \dashleftarrow_{\mathsf{gc}} & (\lambda\alpha.t^\epsilon)^E
\end{array}$$

$\longrightarrow_{\mathsf{d@}}$.

$$\begin{array}{ccc}
(ts)^E & \xrightarrow[\mathsf{d@}]{} & t^E s^E \\
\Big\downarrow{\scriptstyle \mathsf{c}} & & \Big\downarrow{\scriptstyle \Box\mathsf{o}_{\mathsf{l}}} \\
 & & (t' s^E)^{E'} \\
 & & \Big\downarrow{\scriptstyle \Box\mathsf{o}_{\mathsf{r}}} \\
 & & ((t's)^E)^{E'} \\
 & & \Big\downarrow{\scriptstyle \Box\mathsf{m}} \\
(t's)^{E,E'} & \equiv & (t's)^{E,E'}
\end{array}$$

$\longrightarrow_{\mathsf{dF}}$. Same as the case above.

$\longrightarrow_{\mathsf{d}\Box}$. Let F contain n-equations. Then:

$$\begin{array}{ccc}
\langle\langle \alpha \mid E\rangle \mid F\rangle & \xrightarrow[\mathsf{d}\Box]{} & \langle \alpha \mid E^F\rangle \\
\Big\downarrow{\scriptstyle \Box\mathsf{m}} & & \Big\downarrow{\scriptstyle \Box_{\mathsf{e}}} \\
\langle \alpha \mid E, F\rangle & \xrightarrow[\mathsf{c}]{} & \langle \alpha \mid E', F_1, \cdots, F_n\rangle
\end{array}$$

where $\twoheadrightarrow_{\Box_{\mathsf{e}}}$ stands for the reduction relation induced by the box elimination rules and garbage collection.

Summarizing, we have:

$$\begin{array}{ccc}
\cdot & \xrightarrow[\mathsf{d}]{} & \cdot \\
\Big\downarrow{\scriptstyle \Box_{\mathsf{e}}} & & \Big\downarrow{\scriptstyle \Box_{\mathsf{e}}} \\
\cdot & \xrightarrow[\mathsf{c}]{} & \cdot
\end{array} \tag{8.11}$$

From (8.11) and the fact that copying commutes with $\twoheadrightarrow_{\Box_{\mathsf{e}}}$:

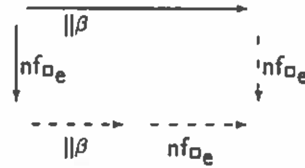
$$\begin{array}{ccc}
\cdot & \xrightarrow[\mathsf{nf_O}]{} & \cdot \\
\Big\downarrow{\scriptstyle \mathsf{nf}_{\Box_{\mathsf{e}}}} & & \Big\downarrow{\scriptstyle \mathsf{nf}_{\Box_{\mathsf{e}}}} \\
\cdot & \xrightarrow[\mathsf{nf}_{\Box_{\mathsf{e}}}\cup\mathsf{cgc}]{} & \cdot
\end{array}$$

where $\twoheadrightarrow_{\mathsf{nf}_{\Box_{\mathsf{e}}}\cup\mathsf{gc}}$ stands for the reduction relation induced by $\longrightarrow_{\mathsf{nf}_{\Box_{\mathsf{e}}}}$ and $\longrightarrow_{\mathsf{cgc}}$.

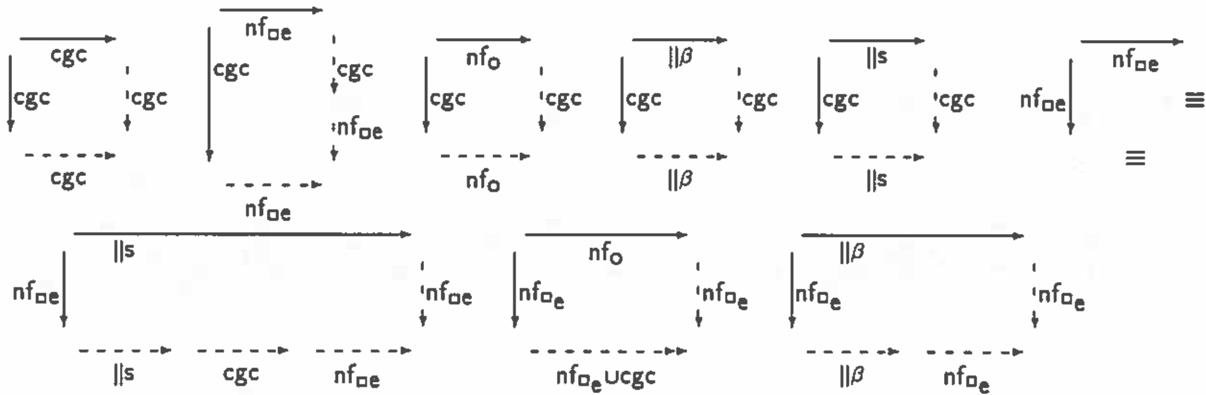
$\longrightarrow_{\mathsf{nf}_{\Box_{\mathsf{e}}}}$ **and** $\longrightarrow_{\|\beta}$. The only interference is caused by $\longrightarrow_{\Box\lambda}$:

$$\begin{array}{ccccc}
(\lambda\alpha.t^{E_1,E})s & & \xrightarrow[\beta]{} & & \langle t^{E_1,E} \mid \alpha = s\rangle \\
\Big\downarrow{\scriptstyle \Box\lambda} & & & & =_{\mathsf{nf}_{\Box_{\mathsf{e}}}} \\
(\lambda\alpha.tE_1)^E s & \xrightarrow[\Box@_{\mathsf{l}}]{} & ((\lambda\alpha.tE_1)s)^E & \xrightarrow[\beta]{} & \langle t^{E_1} \mid \alpha = s\rangle^E
\end{array} .$$

Following a similar argument as in the study of the interaction between $\longrightarrow_{d\lambda}$ and β-reduction we then have the following commuting diagram:



Summarizing, we have the diagrams used in the proof of confluence of $\lambda\phi_1$ and the following ones:



According to the ordering $\mathsf{nf}_{\square_e} < \mathsf{cgc} < \mathsf{nf_o} < \| \beta < \| \mathsf{s}$, the above diagrams are decreasing, and thus by Theorem 8.17 $\lambda\phi_2'$ is confluent. As in the proof of confluence of $\lambda\phi_1$, confluence of $\lambda\phi_2$ follows from the fact that a reduction of $\lambda\phi_2$ is a derived reduction in $\lambda\phi_2'$, and each reduction in $\lambda\phi_2'$ is contained in $\lambda\phi_2$. □

Remark 8.23 If we start from a λ-calculus term such that each λ-abstraction does not have trivial *maximal free expressions* (mfe's), than the $\lambda\phi_2$-calculus is able to simulate Wadsworth's interpreter. The trick is played by the β-rule and the box elimination rules: a redex $(\lambda\alpha.M)A$ will be reduced to $\langle M \mid \alpha = A\rangle$, that is, A is put in the environment, as in [HM76] or, following the terminology of [AKP84], A is "flagged" so that it will not be copied in case the redex is shared. This suggests that in order to avoid the extra complication of detecting mfe's at run time, as in [Wad71], a term can be first pre-processed by well-known techniques [Hug82, Joh85]. Then doing sharing of arguments is enough to capture the amount of sharing offered by Wadsworth's interpreter.

Intermezzo 8.24 We present the system introduced by Rose [Ros92b] in our framework. Rose calls his system $\lambda\mu$, not to be confused with the system of Section 7.1. The set of $\lambda\mu$-terms is defined as follows:

$$\begin{array}{lcl} S & ::= & M^{\mu} \\ M & ::= & \alpha \mid (\lambda\alpha.S) \mid (ST) \\ \mu & ::= & \alpha_1 = S_1, \cdots, \alpha_k = S_k \ . \end{array}$$

S stands for a $\lambda\mu$-term; M, P stand for the λ-component stripped of the substitution; μ, ρ, γ and π range over a sequence of equations. The reduction rules are given in Table 7. $\beta\mu$, μ_2, μ_3 and μ_4 can be simulated in $\lambda\phi_2$ as follows:

$\beta\mu$:

$$\begin{array}{ccc} ((\lambda\alpha.M^{\mu})^{\rho}S)^{\gamma} & \xrightarrow[\beta\mu]{} & \langle M \mid \mu, \rho, \alpha = S, \gamma\rangle \\ \Big\downarrow \mathsf{d}\lambda & & \Big\uparrow \Box\mathsf{m} \\ ((\lambda\alpha.M^{\mu\rho})S)^{\gamma} & \xrightarrow[\beta]{} & \langle M^{\mu\rho} \mid \alpha = s\rangle^{\gamma} \end{array}$$

μ_2:

$$\begin{array}{ccc} \langle \alpha_1 \mid \alpha_1 = M_1^{\mu_1}, \alpha_2 = M_2^{\mu_2}\rangle & \xrightarrow[\mu_2]{} & \langle M_1 \mid \mu_1, \alpha_1 = M_1^{\mu_1}, \alpha_2 = M_2^{\mu_2}\rangle \\ \Big\downarrow \mathsf{es} & & \equiv \\ \langle M_1^{\mu_1} \mid \alpha_1 = M_1^{\mu_1}, \alpha_2 = M_2^{\mu_2}\rangle & \longrightarrow_{\Box\mathsf{m}} & \langle M_1 \mid \mu_1, \alpha_1 = M_1^{\mu_1}, \alpha_2 = M_2^{\mu_2}\rangle \end{array}$$

μ_3:

$$\begin{array}{ccc} (\lambda\alpha.M^{\mu})^{\rho} & \xrightarrow[\mu_3]{} & \langle(\lambda\alpha.M^{\mu,\rho}) \mid \rangle \\ \Big\downarrow \mathsf{d}\lambda & & \Big\downarrow \mathsf{gc} \\ \lambda\alpha.M^{\mu\rho} & \xrightarrow[\Box\mathsf{m}]{} & \lambda\alpha.M^{\mu,\rho} \end{array}$$

μ_4:

$$\begin{array}{ccc} (M^{\mu}P^{\pi})^{\rho} & \xrightarrow[\mu_4]{} & \langle(M^{\mu,\rho}P^{\pi,\rho}) \mid \rangle \\ \Big\downarrow \mathsf{d@} & & \Big\downarrow \mathsf{gc} \\ (M^{\mu})^{\rho}(P^{\pi})^{\rho} & \overset{\text{- - -}}{\twoheadrightarrow}_{\Box\mathsf{m}} & M^{\mu,\rho}P^{\pi,\rho} \end{array}$$

Thus, the main difference between $\lambda\phi_2$ and Rose's calculus concerns μ_1, which is absent in $\lambda\phi_2$. Since in a box construct $\langle \alpha_i \mid \alpha_1 = t_1, \cdots, \alpha_n = t_n\rangle$ the order of the equations is irrelevant there is no need of copying equations in order to bring the equation $\alpha_i = t_i$ in first position. The reason that prevents μ_1 to be simulated in $\lambda\phi_2$ is that μ_1 introduces new cyclic boxes. For example, μ_1 allows the following reduction:

$$\langle \delta_1 \mid \delta = \lambda\alpha.\delta_1(\mathsf{S}\alpha), \delta_1 = \lambda\gamma.\delta(\mathsf{S}\gamma)\rangle \longrightarrow_{\mu_1} \langle \delta_1 \mid \delta_1 = \langle \lambda\gamma.\delta(\mathsf{S}\gamma) \mid \delta = \lambda\alpha.\delta_1(\mathsf{S}\alpha)\rangle\rangle \ .$$

The internal box of the right-hand side term is on a cycle and thus cannot be removed.

We can now extend $\lambda\phi_2$ with term rewriting rules.

Theorem 8.25 *Let R be an orthogonal term rewriting system. Then, $\lambda\phi_2 \cup R$ is confluent.*

$\beta\mu$:		
$((\lambda\alpha.M^{\mu})^{\rho}S)^{\gamma}$	$\longrightarrow$	$\langle M \mid \mu,\rho,\alpha = S,\gamma\rangle$
μ_1 :		
$\langle \alpha \mid \alpha_1 = M_1^{\mu_1},\cdots,\alpha_k = M_k^{\mu_k}\rangle$	$\longrightarrow$	$\langle \alpha \mid \alpha_2 = M_2^{\mu_2,\alpha_1=M_1^{\mu_1}},\cdots,\alpha_k = M_k^{\mu_k,\alpha_1=M_1^{\mu_1}}\rangle$ if $\alpha \neq \alpha_1$ and $k \geq 1$
μ_2 :		
$\langle \alpha_1 \mid \alpha_1 = M_1^{\mu_1},\cdots,\alpha_k = M_k^{\mu_k}\rangle$	$\longrightarrow$	$\langle M_1 \mid \mu_1,\alpha_1 = M_1^{\mu_1},\cdots,\alpha_k = M_k^{\mu_k}\rangle$ with the recursion variables defined in μ_1 not occurring free in $M_i^{\mu_i}, i \geq 2$.
μ_3 :		
$(\lambda\alpha.M^{\mu})^{\rho}$	$\longrightarrow$	$\langle \lambda\alpha.M^{\mu,\rho} \mid \rangle$ if ρ is non-empty
μ_4 :		
$(M^{\mu}P^{\pi})^{\rho}$	$\longrightarrow$	$\langle M^{\mu,\rho}P^{\pi,\rho} \mid \rangle$ if ρ is non-empty

Table 7: Rose's $\lambda\mu$-calculus.

Proof: Following the proof of confluence of $\lambda\phi_2$ we can show the following commuting diagrams:

$$\begin{array}{ccc} \cdot & \xrightarrow{\ \|R\ } & \cdot \\ \downarrow{\scriptstyle \mathsf{nf_o}} & & \downarrow{\scriptstyle \mathsf{nf_o}} \\ \cdot & \overset{\|R}{\dashrightarrow}\ \cdot\ \overset{\mathsf{nf_o}}{\dashrightarrow} & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & \xrightarrow{\ \|R\ } & \cdot \\ \downarrow{\scriptstyle \mathsf{nf_{\Box e}}} & & \downarrow{\scriptstyle \mathsf{nf_{\Box e}}} \\ \cdot & \overset{\|R}{\dashrightarrow}\ \cdot\ \overset{\mathsf{cgc}}{\dashrightarrow}\ \cdot\ \overset{\mathsf{nf_{\Box e}}}{\dashrightarrow} & \cdot \end{array}$$

Where $\longrightarrow_{\|R}$ stands for a complete development of a set of R-redexes. □

We can also extend $\lambda\phi_2$ with orthogonal term graph rewriting. With respect to the term rewriting rules:

$$\mathsf{F}(\alpha) \longrightarrow \mathsf{G}(\alpha,\alpha)$$
$$\mathsf{H}(\alpha) \longrightarrow 1$$

instead of reducing the term $\mathsf{F}(\mathsf{H}(\eta))$ as:

$$\mathsf{F}(\mathsf{H}(\eta)) \longrightarrow \mathsf{G}(\mathsf{H}(\eta),\mathsf{H}(\eta))$$

thus duplicating the redex $\mathsf{H}(\eta)$, we would like to keep the substitution in the environment, as in the reduction below:

$$\mathsf{F}(\mathsf{H}(\eta)) \longrightarrow \langle \mathsf{G}(\alpha,\alpha) \mid \alpha = \mathsf{H}(\eta)\rangle \ .$$

One possibility is to introduce a new notion of reduction. If $l \longrightarrow r$ is a first-order term rewriting rule, and l^{σ} a redex, then we can say:

$$l^{\sigma} \longrightarrow \langle r \mid x_1 = t_1,\cdots,x_n = t_n\rangle \ ,$$

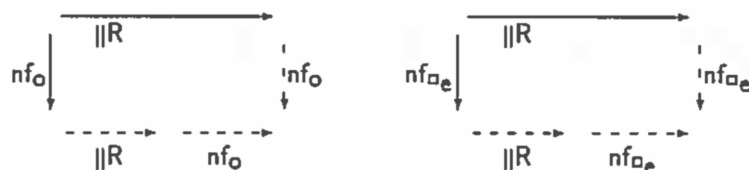
where σ is the mapping $x_1 \mapsto t_1, \cdots, x_n \mapsto t_n$. The alternative we pursue instead is to require the right-hand side of a first-order term rewriting rule to be a $\lambda\phi$-term, which is linear in its free variables. For example, we express the rule $\mathsf{F}(\alpha) \longrightarrow \mathsf{G}(\alpha, \alpha)$ as

$$\mathsf{F}(\alpha) \longrightarrow \langle \mathsf{G}(\delta, \delta) \mid \delta = \alpha \rangle \ .$$

Now rewriting can proceed as in first-order term rewriting.

Theorem 8.26 *Let R be an orthogonal term graph rewriting system. Then, $\lambda\phi_2 \cup R$ is confluent.*

Proof: Since term graph rewriting does not cause a duplication we now have the following commuting diagrams:



□

CONCLUSIONS AND FUTURE DIRECTIONS

The motivation for this work came from the desire of providing a unifying framework for reasoning about execution, compilation, and optimization of programs. To that respect, the goal of this paper is to provide a formal basis that allow one to compute and reason about cyclic structures. This is important since cycles occur at the source level, after parsing, in the intermediate program representation (language), and during program execution. Our next step is to study the effect of different strategies on the time behavior of a program, and to relate them to current optimizations, including loop transformations. We also plan to restrict our calculi so that only values are substitutable (*i.e.*, variables and abstractions), and define call-by-value, call-by-need, and lenient calculi. Show their soundness and completeness.

We perceive our calculi as a first step towards providing a formal tool for reasoning about space, issue of particular importance for lazy languages [Wad87, Spa93], and for reasoning about side-effects operations. More specifically, side-effects operations can be expressed without switching to a new model of computation, such as, the environment model as described by Abelson et al. [AS85]. In fact, there is no need of a new model, by capturing space directly in the program, as also suggested by Meyer [Mey94] and Felleisen [FH92], one can introduce assignment with the substitution model.

Moreover, in order to formalize the compilation and optimization of a program as a rewriting process, we intend to enhance current rewriting technology to cover rules with conditions and priorities. Priorities are associated to the rules in order to impose a certain order, with the intention that a rule which is higher in the order will be the preferred one to apply in case of choice. We will also consider rewriting of disconnected graphs, which, as shown by Pinter et al. [PP94], is useful for detecting parallelizable program structures in sequential programs.

ACKNOWLEDGEMENTS
This work was done at the Department of Computer and Information Science of the University of Oregon, at the Department of Software Technology of CWI, and at the Department of Computer Science of the Free University. Zena Ariola thanks both CWI and the Free University to make her summer visits possible.

The research of the first author has been supported by NSF grant CCR-94-10237. The research of the second author has been partially supported by ESPRIT Basic Research Project 6454-CONFER. Funding for this work has further been provided by the ESPRIT Working Group 6345 Semagraph.

We thank Amr Sabry, Femke van Raamsdonk, and Kristoffer Rose for stimulating discussions about a draft of this paper. The first author thanks Stefan Blom for his constant help and explanations of Vincent's work.

REFERENCES

[AA95] Z. M. Ariola and Arvind. Properties of a first-order functional language with sharing. *Theoretical Computer Science*, 146:69–108, 1995.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[AK95] Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *To appear in Fundamenta Informaticae. Extended version: CWI Report CS-R9552.* 1995.

[AKK+94] Z. M. Ariola, J. W. Klop, J. R. Kennaway, F. J. de Vries, and M. R. Sleep. Syntactic definitions of undefined: On defining the undefined. In *Proc. TACS 94, Sendai, Japan*, 1994.

[AKP84] Arvind, V. Kathail, and K. Pingali. Sharing of computation in functional language implementations. In *Proc. Internal Workshop on High-Level Computer Architecture*, 1984.

[App92] A. Appel. *Compiling with Continuations.* Cambdridge University Press, 1992.

[Ari92] Z. M. Ariola. *An Algebraic Approach to the Compilation and Operational Semantics of Functional Languages with I-structures.* PhD thesis, MIT Technical Report TR-544, 1992.

[Ari93] Z. M. Ariola. Relating graph and term rewriting via Böhm models. In C Kirchner, editor, *Proc. 5th International Conference on Rewriting Techniques and Applications (RTA-93), Montreal, Canada, Springer-Verlag LNCS 690*, pages 183–197, 1993.

[AS85] H. Abelson and J. Sussman, G.J with Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1985.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[BD77] R. Burstall and J. Darlington. A transformation system for developing recursive programs. *JACM, 24:44-67*, January 1977.

[CR90] W. Clinger and J. Rees. Revised[4] report on the algorithmic language Scheme.

Technical Report CIS-TR-90-02, University of Oregon, 1990.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential algorithms, and Functional Programming.* Birkhäuser, 2nd edition, 1993.

[DJ90] N. Dershowitz and J. P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier - The MIT Press, 1990.

[FH92] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 102:235–271, 1992.

[Har86] B. Harper. Introduction to Standard ML. Technical report, ECS-LFCS-86-14, Laboratory for the Foundation of Computer Science, Edinburgh University, 1986.

[HL89] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium, Izu*, 1989.

[HM76] P. Henderson and J. H. Morris. A lazy evaluator. In *Proc. ACM Conference on Principles of Programming Languages*, 1976.

[HPJW+92] P. Hudak, S. Peyton Jones, P. Wadler, B. Boutel, J. Fairbairn, J. Fasel, K. Hammond, J. Hughes, T. Johnsson, D. Kieburtz, R. Nikhil, W. Partain, and J. Peterson. Report on the programming language Haskell. *ACM SIGPLAN Notices*, 27(5):1–64, 1992.

[Hug82] R. J. M. Hughes. Super-combinators. In *Proc. of Lisp and Functional Programming*, 1982.

[JGS93] N. D. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, 1993.

[Joh85] T. Johnsson. Lambda lifting: Transforming programs to recursive equations. In *Proc. ACM Conference on Functional Programming Languages and Computer Architecture, Nancy, France, Springer-Verlag LNCS 201*, 1985.

[KKSdV95a] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Infinitary lambda calculus. In *Proc. Rewriting Techniques and Applications, Kaiserslautern*, 1995.

[KKSdV95b] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Information and Computation*, 119(1), 1995.

[Klo] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Mathematical Centre Tracts 127, CWI, Amsterdam,1980.

[Klo92] J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 1–116. Oxford University Press, 1992.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, 1993. A Collection of Contributions in Honour of Corrado Böhm on the Occasion of his 70th Birthday, guest eds. M. Dezani-Ciancaglini, S. Ronchi Della

Rocca and M. Venturini-Zilli.

[Lau93] J. Launchbury. A natural semantics for lazy evaluation. In *Proc. ACM Conference on Principles of Programming Languages*, pages 144–154, 1993.

[Les94] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. In *Proc. 21st Symposium on Principles of Programming Languages (POPL '94), Portland, Oregon*, pages 60–69, 1994.

[Mey94] A. Meyer. Substitution model for scheme: formal definitions. In *Handout for Computability, Programming and Logic. MIT Laboratory for Computer Science*, 1994.

[Nik91] R. S. Nikhil. Id (version 90.1) reference manual. Technical Report 284-2, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, 1991.

[PJ87] S. L. Peyton Jones. *The implementation of Functional Programming Languages.* Prentice-Hall International, Englewood Cliffs, N.J., 1987.

[PP94] S. S. Pinter and R. Y. Pinter. Program optimization and parallelization. *ACM Transactions on Programming Languages and Systems*, 16(3):305–327, 1994.

[PS92] S. Purushothaman and J. Seaman. An adequate operational semantics of sharing in lazy evaluation. In *4th European Symposium on Programming. Lecture Notes in Computer Science. Springer Verlag, Berlin*, 1992.

[Ros92a] J. G. Rosaz. Taming the Y operator. In *Proceedings of Lisp and Functional Programming*, pages 226–234, 1992.

[Ros92b] K. H. Rose. Explicit cyclic substitutions. In M. Rusinowitch and J. L. Rémy, editors, *Proc. 3rd International Workshop on Conditional Term Rewriting Systems (CTRS-92), Pont-à-Mousson, France, Springer-Verlag LNCS 656*, pages 36–50, 1992.

[Spa93] J. Sparud. Fixing some space leaks without a garbage collection. In *Proc. ACM Conference on Functional Programming Languages and Computer Architecture, Copenhagen*, 1993.

[SPvE93] M. R. Sleep, M. J. Plasmeijer, and M. C. D. J. van Eekelen, editors. *Term Graph Rewriting: Theory and Practice.* John Wiley & Sons, 1993.

[Tur79] D. A. Turner. A new implementation technique for applicative languages. *Software Practice and Experience*, 9:31–49, 1979.

[vO94] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit, 1994.

[Wad71] C. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* 1971. PhD thesis, University of Oxford.

[Wad87] P. Wadler. Fixing some space leaks with a garbage collection. *Software Practice and Experience*, 17(9):595–608, 1987.

[Wad90] P. Wadler. Deforestation: transforming programs to eliminate trees. *TCS, 73:231-248*, 1990.