INFORMATION AND COMPUTATION 139, 154–233 (1997)
ARTICLE NO. IC972651

Lambda Calculus with Explicit Recursion*

Zena M. Ariola

Computer and Information Science Department, University of Oregon, Eugene, Oregon 97401
E-mail: ariola@cs.uoregon.edu

and

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, The Netherlands
E-mail: jwk@cwi.nl

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 and also about compilation and optimizations of programs.

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

* A shorter version of this paper appears in the Proceedings of LICS 94 as "Cyclic Lambda Graph Rewriting" [AK94].

0890-5401/97 $25.00

relies on how faithfully it models reality. In that respect, note that while *cyclic structures* are ubiquitous in a program development system [PJ87], traditional 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]:

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

(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:

$$\mathrm{Y}M \rightarrow M(\mathrm{Y}M).$$

Thus, the above data structure `ones` becomes

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

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

$$\mathrm{Y}(\lambda x.1 : x) \rightarrow (\lambda x.1 : x)(\mathrm{Y}(\lambda x.1 : x)) \rightarrow 1 : (\mathrm{Y}(\lambda x.1 : x)).$$

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 Fig. 1, in which @ stands for application).

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 structure `ones` can be expressed as follows:

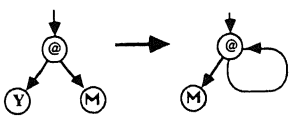


FIG. 1. Cyclic Y-rule.

```
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 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:

```
let fibs=1 : sum fibs (0 : fibs)
                 ρ1
    sum = \x y -> (head x+head y) : sum (tail x) (tail y)
                                          ρ2
in fibs
```

(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 Fig. 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

[1] Rosaz [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.

would like to reduce the redexes (i.e., reducible expressions) ρ_1 and ρ_2 in the Fibonacci program above. These redexes are indicated with an arrow in Fig. 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, since cyclic structures are extensively used by implementors and compiler writers it is important to provide an abstract framework that allows one to reason about them. This paper provides such a framework in the context of λ-calculus and first-order rewriting. The paper is organized as follows. We start, in

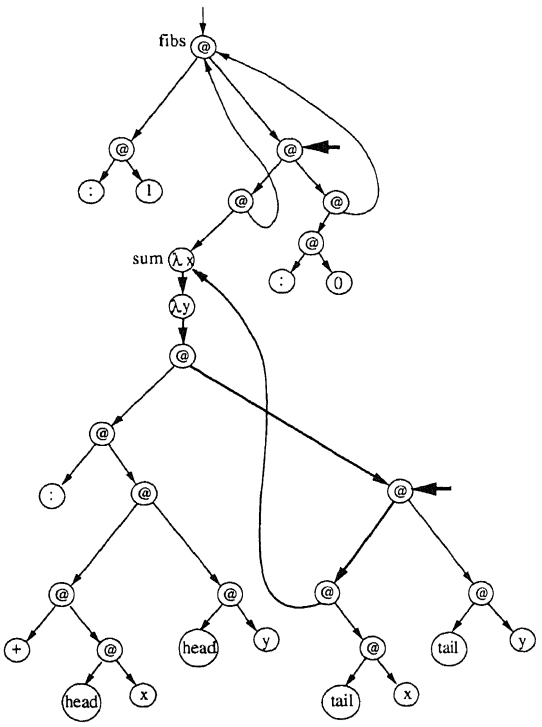


FIG. 2. Cyclic lambda graph for computing the sequence of Fibonacci numbers.

Section 1, by introducing our approach to cycles that is based on systems of recursion equations. Until Section 9, 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. 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 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 show soundness of $\lambda\Theta$ with respect to the infinitary lambda calculus. In Section 9, 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, and term graph rewriting. In Section 10, we discuss previous work. In particular, we relate our approach to Rose's system [Ros92b] and to the framework based on the interaction nets of Lafont [Laf90]. 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 1–8) 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 $\alpha, \delta, \ldots$, for recursion variables. For the time being, variables bound by λ are denoted by $x, y, z, \ldots$. Note that the infinite tree unwinding of the last recursion system is not a Böhm-tree, as it contains many β-redexes.

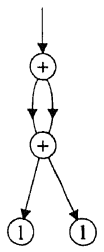


FIG. 3. Horizontal sharing.

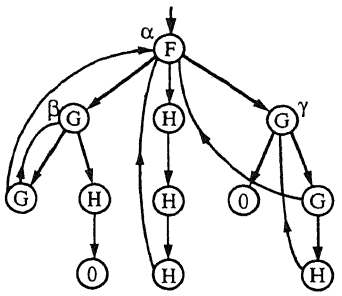


FIG. 4. Vertical sharing.

These systems of recursion equations allow us to express *horizontal sharing*, i.e., sharing as in a *dag* (see Fig. 3), as opposed to the *vertical sharing* shown in the examples above. More precisely, we say that a graph *has only vertical sharing* if the graph can be partitioned into a tree and a set of edges with the property that either begin and end nodes are identical, or the end node is an ancestor (in the tree) of the begin node. Equivalently, a graph has only vertical sharing if there are no two different acyclic paths starting from the root to the same node (see Fig. 4). 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 Fig. 5 is not expressible, as the intuitive representation

$$\alpha = \lambda x.\ +(y, y),\ y = +(1, x),$$

is not correct. This limitation will be removed in the second part of the paper, Section 9, in which we introduce a framework with nested recursion equations. We restrict ourselves to systems without nesting since interesting observations can already be made.

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 the presentation of a recursion system, it is understood that the first (or top-most) 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

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

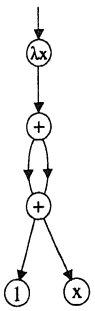


FIG. 5. Lambda body with horizontal sharing.

The order of the equations in the body of the $\langle \mid \rangle$ 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 9, we study systems of recursion equations of the form

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

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

1.1. Correspondence with Graphs

It is straightforward to assign actual graphs to the recursion systems as introduced above. Several examples will be presented later. One feature should be mentioned explicitly: the nodes of the graph contain first-order operators (F), or application (@), or λx, or a variable $x, y, z, ...$ Other than that, a node may have a *name* $\alpha, \delta, \gamma, ...$ These correspond to the recursion variables in the recursion

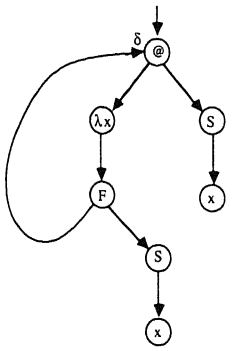


FIG. 6. Cyclic lambda graph corresponding to $\delta = (\lambda x.\mathsf{F}(\delta, \mathsf{S}x))(\mathsf{S}x)$.

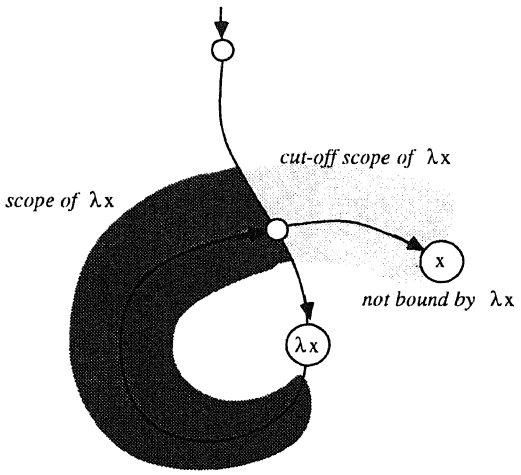


FIG. 7. Scope cut-off phenomenon.

system. Note that unnamed nodes may also 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.

1.2. Free and Bound Variables

The notion of a variable (x, y, ...) 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 Fig. 6. 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 Fig. 7). This is consistent with other ways of presenting the cyclic λ-graph of Fig. 6. 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.

The same scope cut-off phenomenon occurs in the system

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

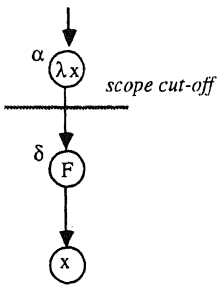


FIG. 8. Cyclic lambda graph corresponding to $\alpha = \lambda x.\delta,\ \delta = \mathsf{F}x$.

which is displayed in Fig. 8; 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 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:

$$\begin{aligned}
&\langle \alpha \mid \alpha = \underline{(\lambda x.\delta x x)\,\alpha},\ \delta = (\lambda y.\alpha y)\,\delta \rangle && \to_\beta \\
&\langle \alpha \mid \alpha = \delta\alpha\alpha,\ \delta = \underline{(\lambda y.\alpha y)\,\delta} \rangle && \to_\beta \\
&\langle \alpha \mid \alpha = \delta\alpha\alpha,\ \delta = \alpha\delta \rangle.
\end{aligned}$$

Here, there is no problem. We call $(\lambda x.\delta x x)\,\alpha$ 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 $\rightarrow_s$; we will occasionally underline the variable we substitute for. Thus:

$$\begin{aligned}
&\langle \alpha \mid \alpha = \lambda x.\underline{\delta}(\mathsf{S}x),\ \delta = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_s \\
&\langle \alpha \mid \alpha = \lambda x.\underline{(\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x)},\ \delta = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_\beta \\
&\langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x)),\ \delta = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_{gc} \\
&\langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}(\mathsf{S}x))\rangle.
\end{aligned}$$

In the last step, we have applied garbage collection (written as $\rightarrow_{gc}$) 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:

$$\begin{aligned}
&\langle \alpha \mid \alpha = \mathsf{F}(\underline{\delta}\mathsf{0}, \delta\mathsf{1}),\ \delta = \lambda x.x\rangle && \rightarrow_s \\
&\langle \alpha \mid \alpha = \mathsf{F}(\underline{(\lambda x.x)\ \mathsf{0}}, \delta\mathsf{1}),\ \delta = \lambda x.x\rangle && \rightarrow_\beta \\
&\langle \alpha \mid \alpha = \mathsf{F}(\mathsf{0}, \delta\mathsf{1}),\ \delta = \lambda x.x\rangle.
\end{aligned}$$

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

2.1. *The Collapse Problem*

In orthogonal term graph rewriting (rewriting with an orthogonal first-order term rewriting system, admitting graphs with horizontal and vertical sharing) 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 $\mathsf{I}(x) \rightarrow x$. 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 $\mathsf{J}(x) \rightarrow x$. 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 a "very undefined" kind of expression; this is a special case of expressions being undefined by lack of a head normal form. We capture this fact by rewriting $\langle \alpha \mid \alpha = \alpha\rangle$ to a new object, that we will call • (black hole). For a comparison of notions of undefinedness in orthogonal term (graph) rewriting see [AKK+94].

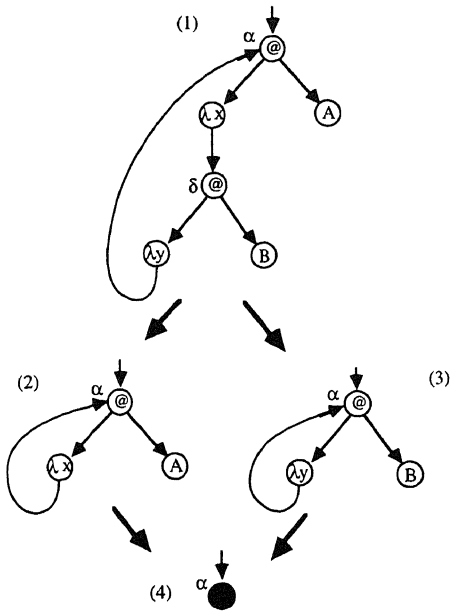


FIG. 9. Reductions to black hole.

Also, in the present setting, • arises as a result of reduction; e.g., consider the λ-graph (see Fig. 9(1))

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

Contracting the ρ-redex yields

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

which is equivalent to (see Fig. 9(3))

$$\alpha = (\lambda y.\alpha)\, B.$$

Contracting the τ-redex yields

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

which is equivalent to (see Fig. 9(2))

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

Both contracted graphs yield after one more reduction $\alpha = \alpha$, and this rewrites to $\alpha = \bullet$ (see Fig. 9(4)). Note that mutual vacuous dependencies of recursion variables also rewrite to •; e.g., $\langle \alpha \mid \alpha = \delta,\ \delta = \alpha \rangle \twoheadrightarrow \bullet$. Or, inside a system,

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

3. THE λΘ-CALCULUS

Here we present the $\lambda\Theta$-calculus, which formalizes the naive way of reducing possibly cyclic redexes introduced so far. Notation: We assume that F^n belongs to a first-order signature. The metavariables E, E' 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. $C[\,\Box\,]$ represents a λ-calculus context with one hole $\Box$. A system of equations E' is orthogonal to a system E or to a variable α if the recursion variables of E' (i.e., the variables that occur as the left-hand side of an equation in E') do not intersect with the set of free variables of E and α.

DEFINITION 3.1. The following clauses define the syntax and basic reduction axioms of the $\lambda\Theta$-calculus.

SYNTAX:

$$g ::= \alpha_1 = M_1, \ldots, \alpha_n = M_n$$

$$M ::= x \mid \mathsf{F}^n(M_1, \ldots, M_n) \mid \lambda x.M \mid MM$$

REDUCTION AXIOMS:

β-rule:

$$(\lambda x.M)\, N \quad \rightarrow_\beta \quad M[x := N]$$

Substitution:

$$\langle \alpha \mid \gamma = C[\delta], \delta = M, E \rangle \quad \rightarrow_s \quad \langle \alpha \mid \gamma = C[M], \delta = M, E \rangle$$

Black hole:

$$\langle \alpha \mid \gamma = \gamma, E \rangle \quad \rightarrow_\bullet \quad \langle \alpha \mid \gamma = \bullet, E \rangle$$

Copying:

$$\langle \alpha \mid E \rangle \quad \rightarrow_c \quad \langle \alpha' \mid E' \rangle$$

if there exists a variable mapping σ, $\langle \alpha' \mid E' \rangle^\sigma \equiv \langle \alpha \mid E \rangle$

Naming:

$$\langle \alpha \mid \gamma = C[M], E \rangle \quad \rightarrow_n \quad \langle \alpha \mid \gamma = C[\delta], \delta = M, E \rangle$$

if the free variables of M do not occur bound in $C[M]$ and M is not a variable

Garbage collection:

$$\langle \alpha \mid E, E' \rangle \quad \rightarrow_{gc} \quad \langle \alpha \mid E \rangle$$

if E' is non-empty and orthogonal to E and α

In the *Substitution* rule, the equations $\gamma = C[\delta]$ and $\delta = M$ can overlap as in the substitution step

$$\langle \alpha \mid \alpha = \lambda x.x\alpha \rangle \to_s \langle \alpha \mid \alpha = \lambda x.x(\lambda x.x\alpha) \rangle,$$

in which both δ and γ are instantiated to α. The operation of copying differs from substitution in the sense that copying never gets rid of 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 $g_1^\sigma \equiv g$, leaving the free recursion variables of g_1 unchanged. For example,

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

where the variable mapping σ is as follows: α, α' are mapped to α, and γ, γ' are mapped to γ. (See [AK96] for a thorough discussion of copying and its properties.) The proviso for the operation of naming, which is written as $\to_n$, is to forbid reductions of the form

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

in which the underlined x gets out of scope.

To understand why we also admit, in addition to substitution, the operations of copying and naming, we make an excursion into the first-order case. Substitution by itself causes non-confluence already 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; they are *out of sync*, that is, 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 [AK96], confluence of substitution and naming is guaranteed if the system also contains the operation of copying. Thus, in analogy with the first-order case, we consider after substitution 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 3.2. 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

$$\alpha = \lambda y.\alpha 0 \;\to_{\mathrm{n}}\; \begin{matrix}\alpha = \lambda y.\delta,\\ \delta = \alpha 0\end{matrix} \;\to_{\mathrm{s}}\; \begin{matrix}\alpha = \lambda y.\delta,\\ \delta = (\lambda y.\delta)\,0\end{matrix} \;\to_{\beta}\; \begin{matrix}\alpha = \lambda y.\delta,\\ \delta = \delta\end{matrix} \twoheadrightarrow \alpha = \lambda y.\bullet.$$

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.

This example points out that in order to describe common program manipulations, as the one described above, it is necessary to precisely delimit the body of a lambda abstraction, thus indicating how much to copy once the lambda is applied. In our simple framework, all unnamed nodes reachable from a lambda-node constitute its body.

4. A COUNTEREXAMPLE TO CONFLUENCE OF λΘ

Consider the reductions (displayed in Fig. 10):

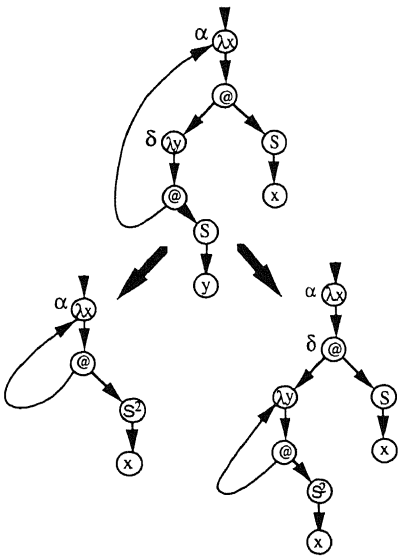
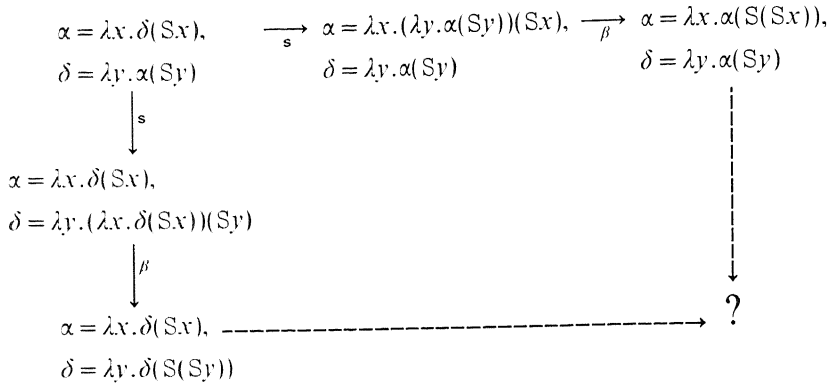


FIG. 10. Failure of confluence.



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

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

are irreversibly separated with respect to any set of operations on λ-graphs that is "sound" in a sense that we will elaborate in Section 8.

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

$$\#\ \text{let rec odd} = \text{fun } x \to \text{if } x = 0 \text{ then false else } \underbrace{\text{even}(x-1)}_{\rho_1}$$

$$\text{and even} = \text{fun } x \to \text{if } x = 0 \text{ then true else } \underbrace{\text{odd}(x-1)}_{\rho_2};;$$

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 gets rid of the definition of even.

4.1. Analysis of the Counterexample

The above counterexample is a counterexample not only to confluence, but also 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 on the converging sides, one has to contract the descendants (residuals) of the redexes contracted on 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 the result

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

where $\mathsf{S}^2(x)$ stands for $\mathsf{S}(\mathsf{S}(x))$.

In the other direction, we contract redex 2, after making it explicit:

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

Contraction of the redex 2 yields

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

So, in analogy with 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:

$$\begin{array}{lll} \alpha = \lambda x.[\delta(\mathsf{S}x)]_1,\ \delta = \lambda y.[\alpha(\mathsf{S}y)]_2 & \longrightarrow_1 & \alpha = \lambda x.[\alpha(\mathsf{S}^2x)]_2 \\ \Big\downarrow{\scriptstyle 2} & & \Big\downarrow{\scriptstyle 2} \\ \alpha = \lambda x.[\delta(\mathsf{S}x)]_1, \delta = \lambda y.[\delta(\mathsf{S}^2y)]_1 & \longrightarrow_1 & ? \end{array}$$

Now the reason for 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. For greater ease in parsing the following expressions, let us use underlining instead of $[\ ]_2$ to keep track of implicit or explicit redexes, so

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

is now

$$\alpha = \lambda x.\underline{\alpha(S^2 x)}.$$

We claim that this singleton set of underlined redexes cannot be completely developed, as the analogy with λ-calculus suggests we ought to do. Indeed, it is easily seen that no succession of $\to_s$ or $\to_\beta$ in whatever order will be able to remove all underlining, using obvious rules for underlining:

$$\begin{aligned}
&\alpha = \lambda x.\underline{\alpha(S^2 x)} && \to_s \\
&\alpha = \lambda x.\underline{(\lambda x.\underline{\alpha(S^2 x)})(S^2 x)} && \to_\beta \\
&\alpha = \lambda x.\underline{\alpha(S^4 x)} && \to_s \\
&\alpha = \lambda x.\underline{(\lambda x.\underline{\alpha(S^4 x)})(S^4 x)} \cdots
\end{aligned}$$

(also applying $\to_s$ on the second expression does not bring us further).

This elaboration is meant to give an intuition as to why confluence fails—of course it does not constitute a proof of that failure.

4.2. *Another Analysis of the Counterexample*

Consider the following abstract reduction system, with 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:

$$\begin{aligned}
\{n\} &\to \{2n\} \\
\{n\} &\to (n, n) \\
\{n\} &\to [n, n] \\
(n, m) &\to (n + m, m) \\
(n, m) &\to (n, 2m) \\
[n, m] &\to \{n + m\} \\
[n, m] &\to (n, n + m).
\end{aligned}$$

We claim that these are not confluent. Proof: $[1, 1] \to \{2\}$ and $[1, 1] \to (1, 2)$. 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:

$$\{n\}: \quad \alpha = \lambda x.\alpha(\mathsf{S}^n x)$$

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

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

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,

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

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:

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

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

This phenomenon may be thought to be dependent on 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].

We first emphasize the difference between convergent and strongly convergent reductions. In short, a strongly convergent reduction is such that the prefix of the term where no reduction occurs is increasing (see Fig. 11), that is, the depth of contracted redexes tends to infinity.

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 possibly infinite λ-tree, and let S be a set (possibly infinite) of redexes in M.

(i) A *development* of S is a reduction, possibly infinite, in which only descendants of members of S are contracted.

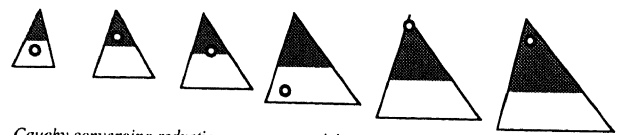
(ii) A *complete development* of S is a development which is strongly convergent and after which no descendant of a redex of S is left.

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 this for the infinitary λ-calculus, since it admits infinitely many redexes to be developed. But there is an analogous statement, that is, 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.

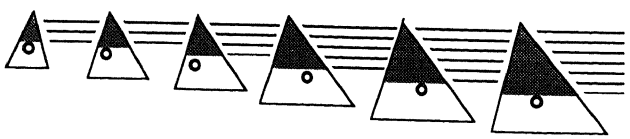
Consider the infinite unwinding of the term

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

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



Cauchy converging reduction sequence: activity may occur everywhere



Strongly converging reduction sequence, with descendant relations

FIG. 11. Converging and strongly converging reduction sequences.

of S_1 and S_2 is shown in Fig. 12 and 13, 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 a unary operator $\underline{n}$ for every $n \geqslant 0$. There are infinitely many rewrite rules:

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

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

$$\begin{array}{llllllllllll} \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots \to \\ 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots & \to \\ 2 & & 2 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & \cdots & \to_{\omega} \\ 2 & & 2 & & 2 & & 2 & & 2 & & \cdots & \end{array}$$

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 Fig. 12, whose leftmost infinite tree is represented as the infinite term 1 1 1 $\cdots$ (i.e., at each level the tree contains one symbol S). The complete development of the redexes marked with **2** in Fig. 13 corresponds to the reduction of the redexes at odd positions, yielding

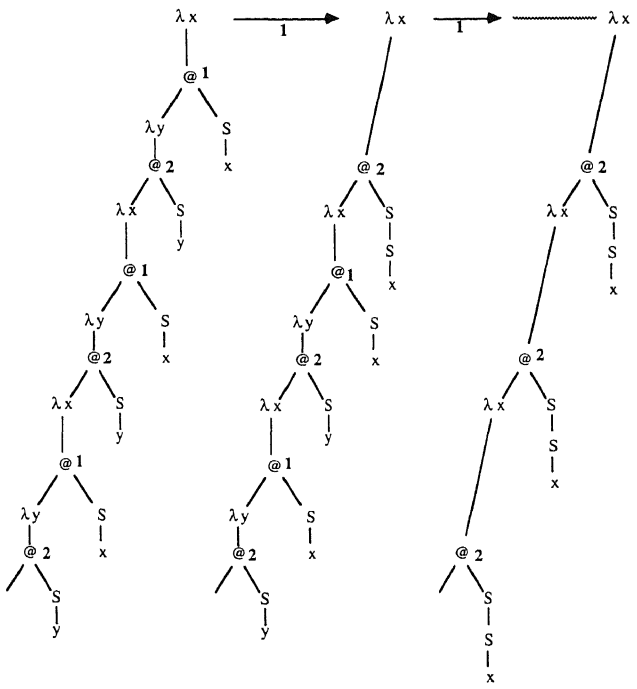


FIG. 12. Complete development of the redexes marked as **1**.

6. REGULAR DEVELOPMENTS AND ANOTHER COUNTEREXAMPLE

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 weak head normal form. Or, equivalently, that confluence can be restored by equating all $\lambda\Theta$-expressions that have no weak 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.

An example of such an expression is

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

Indeed, it is easily verified that the corresponding infinite term reduces to the infinitary normal form $\lambda x.(FG)^{\omega}$, independent of the order of reduction of the redexes of the form yx and αy. Establishing that this is indeed a counterexample to confluence in $\lambda\Theta$ can be done by a reasoning similar to that for the counterexample in Section 4.

A more interesting counterexample is as follows. 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

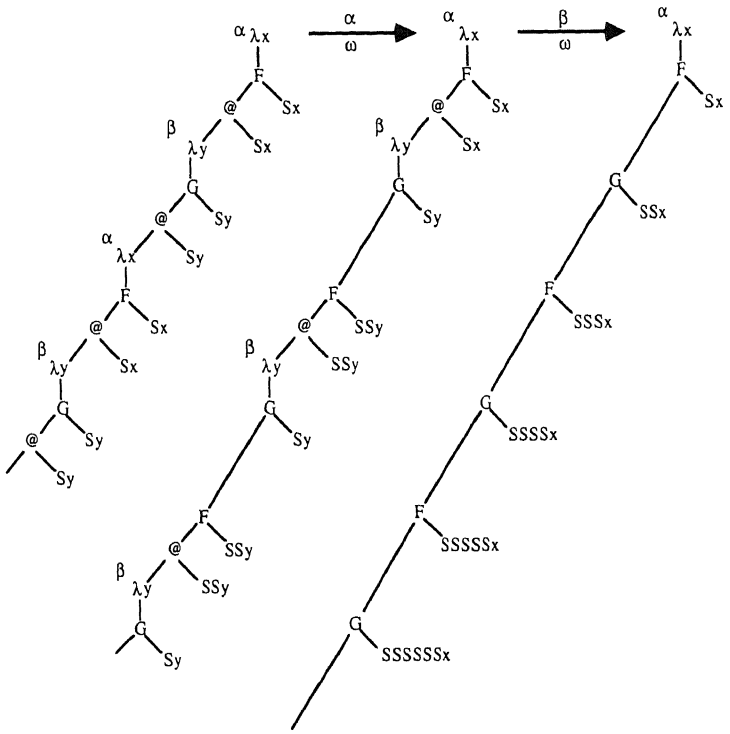


FIG. 14. Infinite reduction yielding a non-regular tree.

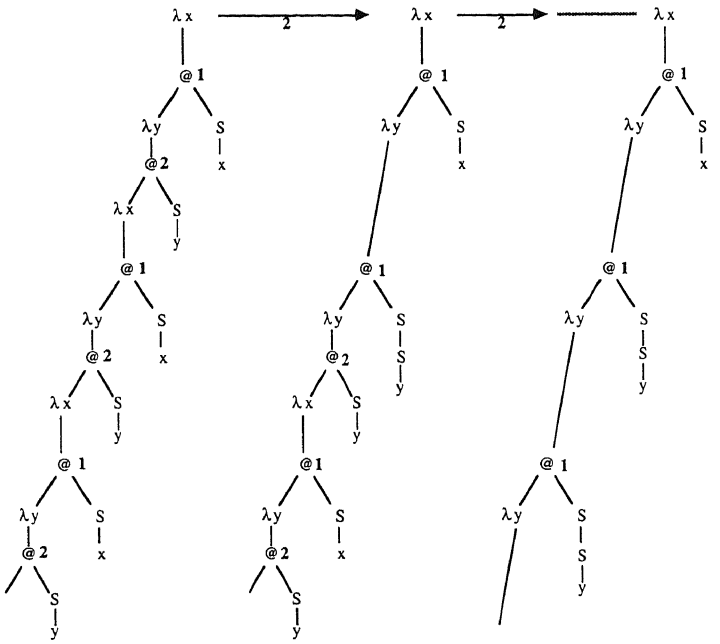


FIG. 13. Complete development of the redexes marked as **2**.

$$
\begin{aligned}
&1\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \cdots\ \rightarrow\\
&1\quad 2\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \cdots\ \rightarrow\\
&1\quad 2\quad 2\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 1\ \cdots\ \rightarrow_{\omega}\\
&1\quad 2\quad 2\quad 2\quad 2\quad 2\ \cdots
\end{aligned}
$$

Now it is clear that the two infinite terms $2\ 2\ 2\ 2\ \cdots$ and $1\ 2\ 2\ 2\ 2\ \cdots$ have no common reduct. A side result of this example is that for confluent and terminating TRS's the generalization to infinitary rewriting does not work out well; apparently the orthogonality condition is needed.

Identifying the larger class of terms without weak head normal form does restore confluence for the infinitary λ-calculus. A term M has a weak head normal form if it reduces to some term of the form $xN_1 \cdots N_n$ $(n \geqslant 0)$ or $\lambda x.N$.

THEOREM 5.2. *The infinitary lambda calculus extended with the rule (called Ω-rule)*

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

is infinitary confluent.

Proof. See [KKSdV95a], in which the infinitary calculus referred to in this paper is called the 111-infinitary calculus. ∎

Remarkably, this is not the case in λ-graph rewriting, as discussed in the next section.

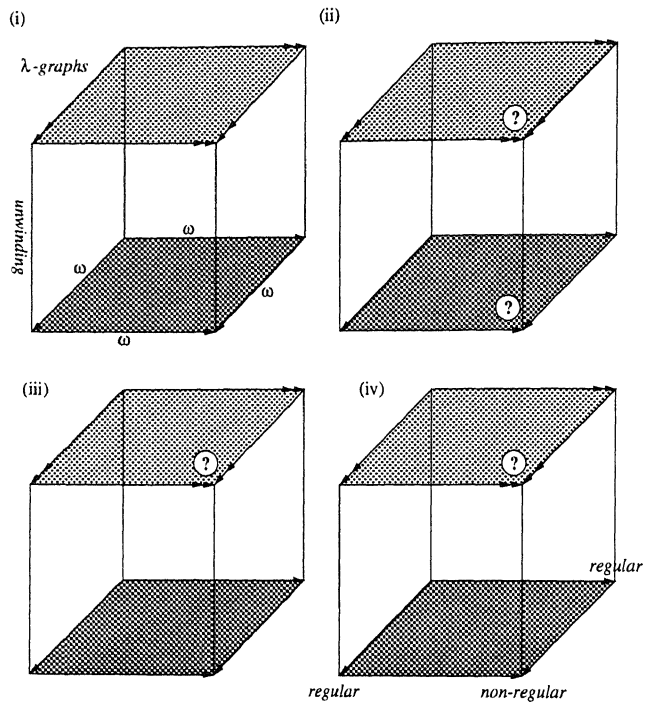


FIG. 15. Relation between λ-graph and λ-tree reductions.

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

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

Unwinding these recursion equations yields the infinite λ-tree in the leftmost corner of Fig. 14. 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 β yields a regular tree. But developing both sets of redexes yields a non-regular tree, namely, the rightmost one of Fig. 14. To see that we have indeed another counterexample to confluence in $\lambda\Theta$, we reason as follows, using the soundness of $\lambda\Theta$ with respect to the infinitary calculus (shown in Section 8). Let M be the initial (leftmost) infinite term in Fig. 14. Let M_α be the middle term in that figure, arising after developing all α-redexes in M. Likewise M_β is the term arising from M after developing all redexes marked β in M; this term is not shown in the figure. Finally, let $M_{\alpha,\beta}$ be the term arising from developing both families of redexes, the redexes marked α as well as the redexes marked β. This is the common reduct in the infinitary calculus of M_α and M_β. Now we claim that in fact $M_{\alpha,\beta}$ is the *only* common reduct of M_α and M_β. Establishing this is a matter of routine which we omit. It follows that in the finite graph calculus $\lambda\Theta$ there cannot be a common reduct for the two expressions arising from the recursion system under consideration after executing the redex $\beta(Sx)$ on the one hand and $\alpha(Sy)$ on the other hand. For these two expressions unwind to M_β and M_α, respectively. Now if the two expressions

would have a common reduct, say C, in $\lambda\Theta$, then by its soundness we would have in the infinitary calculus a common reduct of M_α and M_β, namely the unwinding of C. But this would be a regular term, as C is a finite expression in $\lambda\Theta$, in contradiction with the claim above that the irregular term $M_{\alpha,\beta}$ is the only common reduct of M_α and M_β.

Summarizing, we have the situations in Fig. 15, where for each cube 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 15(i) displays the "normal" situation. Figure 15(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 15(iii) refers to the first counterexample in the present section. Figure 15(iv) refers to the second counterexample in this section involving developments to non-regular infinite terms.

7. NOTIONS OF SUBSTITUTION

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 system as follows:

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

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, 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 $\rightarrow_{\mathsf{as}}$), consists of defining an order on the nodes of a graph, or equivalently on the recursion variables (see Fig. 16), and then allowing substitution upward 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:

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

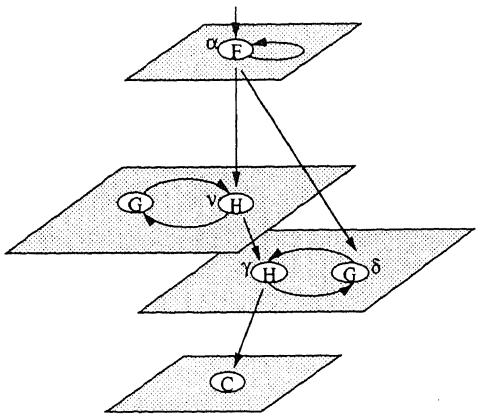


FIG. 16. Ordering among recursion variables.

In $C[\delta]$ just one occurrence of δ is displayed and replaced by M. So in Fig. 16, displaying the system

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

the only $\rightarrow_{\mathsf{as}}$-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 next.

DEFINITION 7.1. The following clauses define the syntax and basic reduction axioms of the $\lambda\Phi$-calculus.

SYNTAX:

$$g ::= \alpha_1 = M_1, \ldots, \alpha_n = M_n$$
$$M ::= x \mid \mathsf{F}^n(M_1, \ldots, M_n) \mid \lambda x.M \mid MM$$

REDUCTION AXIOMS:

β-rule:

$$(\lambda x.M)\, N \qquad \rightarrow_\beta \quad M[x := N]$$

Acyclic substitution:

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

Fact 7.2. Acyclic substitution is non-terminating; e.g.,

$$\alpha = \mathsf{F}\gamma,\ \gamma = \mathsf{G}\gamma \rightarrow_{\mathsf{as}} \alpha = \mathsf{FG}\gamma,\ \gamma = \mathsf{G}\gamma \rightarrow_{\mathsf{as}} \alpha = \mathsf{FGG}\gamma,\ \gamma = \mathsf{G}\gamma \rightarrow_{\mathsf{as}} \cdots.$$

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 $\lambda\Phi\underline{as}$. 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 $\lambda\Phi\underline{as}$ is

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

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 7.3. *Let $g \to_{\underline{as}} g_1$ by reducing redex $\underline{\delta}$ and $g \to_{\underline{as}} g_2$ 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 $g \to_{\underline{as}} g_1$ by substituting for $\underline{\delta}$ in α, and $g \to_{\underline{as}} g_2$ 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 Fig. 16). Note that $\delta = \eta$ and $\gamma = \alpha$ are not simultaneously possible. Let us assume $\delta = \eta$. We have

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

∎

LEMMA 7.4. *$\twoheadrightarrow_{\underline{as}}$ 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

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

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

THEOREM 7.5. *Acyclic substitution is confluent.*

Proof. As in [Bar84, Klo], confluence follows from Lemmas 7.3 and 7.4 by applying the complete development method, which consists of defining a new reduction relation with the same transitive closure as $\twoheadrightarrow_{\mathsf{as}}$ and prove that it satisfies the diamond property. ∎

Next, we prove that acyclic substitution combined with β-reduction is confluent. We thus extend $\lambda\Phi\underline{as}$ by allowing the underlining of λ's that constitute the operator part of a β-redex. The new β-rule becomes $(\underline{\lambda}x.M)\,N \to_{\beta} M[x := N]$. The combination of $\underline{\mathsf{as}}$ and $\underline{\beta}$ is written as $\twoheadrightarrow_{\underline{\mathsf{as}\beta}}$. The new calculus is called $\underline{\lambda\Phi}$ and is summarized next.

DEFINITION 7.6. The following clauses define the syntax and basic reduction axioms of the $\underline{\lambda\Phi}$-calculus.

SYNTAX:

$$g ::= \alpha_1 = M_1, ..., \alpha_n = M_n$$

$$M ::= x \mid \underline{x} \mid \mathsf{F}^n(M_1, ..., M_n) \mid \lambda x.M \mid MM \mid (\underline{\lambda}x.M)\,M$$

REDUCTION AXIOMS:

β-rule:

$$(\underline{\lambda}x.M)\,N \qquad \to_{\underline{\beta}} \quad M[x := N]$$

Acyclic substitution:

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

We start by showing that $\twoheadrightarrow_{\underline{as}\beta}$ 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 variables) 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 7.7. 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:

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

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

$$\alpha = (\underline{\lambda}x.x^{9})(\mathsf{G}\underline{\gamma}^{7}\underline{\gamma}^{8}),\ \gamma = \delta^{5}\delta^{1} \qquad \text{and} \qquad \alpha = (\underline{\lambda}x.x^{9})(\mathsf{G}\underline{\gamma}^{6}\underline{\gamma}^{1}),\ \gamma = \delta^{5}\delta^{2}$$

violate the conditions (i) and (ii), respectively, of Definition 7.7.

PROPOSITION 7.8. *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 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 on 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

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

g has the dwp. ∎

PROPOSITION 7.9. *If* $g \to_{\underline{as\beta}} g_1$ *and* g *has* dwp *then*

$$|g| > |g_1|.$$

Proof. Follows from the fact that $2^m > 2^{m-1} + 2^{m-2} + \cdots 2 + 1$. ∎

PROPOSITION 7.10. *Let* $g \to_{\underline{as\beta}}$, *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

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

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

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

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 7.11. $\twoheadrightarrow_{\underline{as\beta}}$ *is strongly normalizing.*

Proof. From Propositions 7.9 and 7.10. ∎

LEMMA 7.12. *Acyclic substitution commutes with* β.

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

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

By cases where the substitution occurs:

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

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

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

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

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

$$
\begin{array}{lll}
\delta = C_3[\underline{\alpha}], & \xrightarrow[\beta]{} & \delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.\, C_1[\underline{\gamma}])\, C_2[\underline{\eta}], & & \alpha = (C_1[\underline{\gamma}])[x := C_2[\underline{\eta}]], \\
\gamma = M, & & \gamma = M, \\
\eta = N & & \eta = N \\
\quad \Big\downarrow \underline{\text{as}} & & \qquad \Big\downarrow \underline{\text{as}} \\
\delta = C_3[\underline{\alpha}], & \xrightarrow[\beta]{} & \delta = C_3[\underline{\alpha}], \\
\alpha = (\underline{\lambda} x.\, C_1[\underline{\gamma}])\, C_2[N], & & \alpha = (C_1[\underline{\gamma}])[x := C_2[N]], \\
\gamma = M, & & \gamma = M, \\
\eta = N & & \eta = N
\end{array}
$$

∎

THEOREM 7.13. *$\lambda\Phi$ is confluent.*

Proof. From the previous lemma and Hindley–Rosen's lemma. ∎

Confluence of $\lambda\Phi$ guarantees that the lack of confluence of $\lambda\Theta$ does not impair its correctness, as shown in the next section. Moreover, it also allows us to precisely identify which redexes cause confluence to fail, namely the spine-cyclic redexes. A β-redex is *spine-cyclic* when its root and the λ-node lie on the same cycle (see Fig. 17). Otherwise, the redex is *spine-acyclic*. In a spine-acyclic redex the root and the λ-node may be cyclic. Reduction of explicit spine-cyclic redexes, such as the β-redex in the equation $\gamma = (\lambda x.\, C[\gamma])\, M$, does not introduce any problem. Summarizing, Theorem 7.13 says that reduction of implicit and explicit spine-acyclic redexes and explicit spine-cyclic redexes is confluent, since their reduction involves acyclic substitution only.

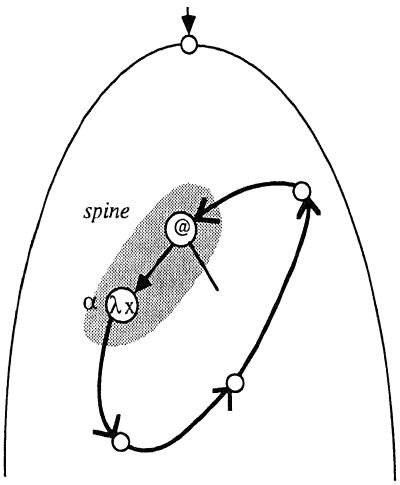


FIG. 17. Cycle through a spine.

An example of a spine-acyclic redex is the topmost β-redex of Fig. 2. The lower β-redex 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 a substitution that is not acyclic is needed to make it explicit. Implicit spine-cyclic redexes can be made explicit by first applying the operation of copying, which allows us to unwind a cycle without losing any name. For example, if we want to reduce the underlined implicit spine-cyclic 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:

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

The system so obtained 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 7.14. Another notion of substitution that guarantees confluence is the *parallel substitution* ($\to_{\mathsf{ps}}$), which consists of substituting at once for all the recursion variables:

$$\alpha_1 = M_1, \ldots, \alpha_n = M_n \to_{\mathsf{ps}} \alpha_1 = M_1[\vec{\alpha}_n := \vec{M}_n], \ldots, \alpha_n = M_n[\vec{\alpha}_n := \vec{M}_n].$$

For example, we have

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

This notion is interesting since it allows us to remove the definition of δ. However, we do not pursue the study of this notion since it does not underlie common program transformations.

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:

$$\mu\colon \mu x.Z(x) \to Z(\mu x.Z(x)).$$

Here we use the notation used for higher-order term rewriting by means of combinatory reduction systems (CRSs), as in [KvOvR93]. Usually this rewriting rule is presented as $\mu x.Z \to Z[x := \mu x.Z]$. The $\lambda\mu$-calculus already offers a form of cyclic λ-graph rewriting (see Fig. 18); 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

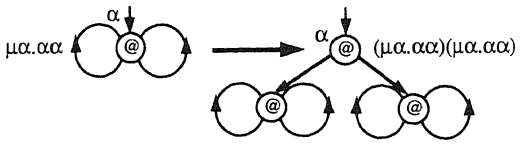


FIG. 18. Reduction of $\mu\alpha.\alpha\alpha$.

system, is confluent. Translating the $\lambda\Theta$-counterexample to confluence, presented in Section 4, into $\lambda\mu$ is instructive. The uppermost cyclic graph of Fig. 10 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 have to apply the μ-rule twice, obtaining

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

The above reduction is displayed in Fig. 19. In Fig. 20 we display one step of substitution. Comparing the middle graph of Fig. 19 and the rightmost graph of Fig. 20 we see that the substitution operation embodied in the μ-rule is much more complex than the unrestricted version of $\lambda\Theta$, since it involves making an entire copy

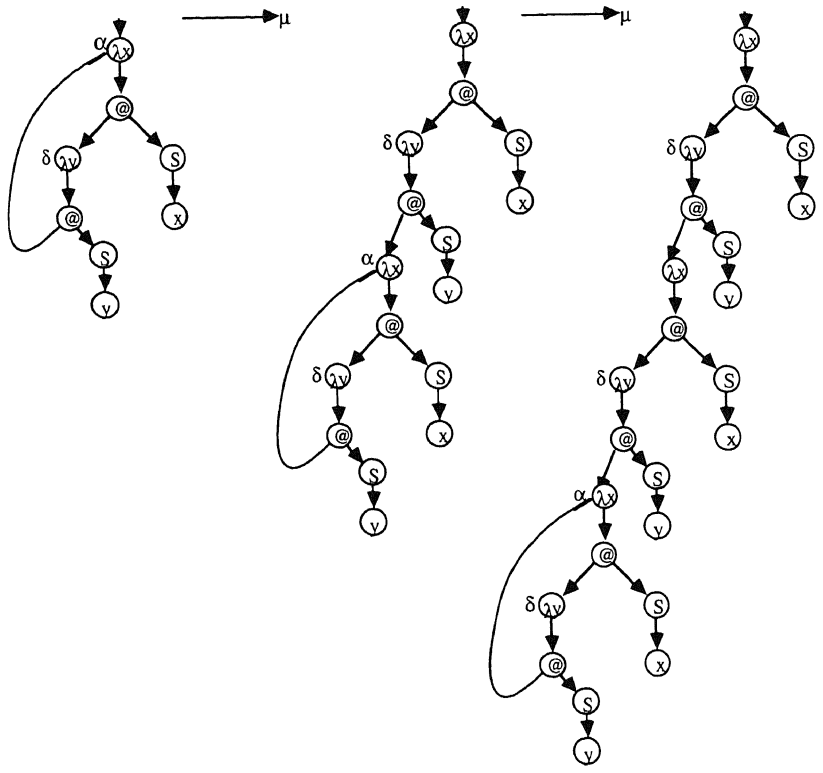


FIG. 19. μ-reduction of $\mu\alpha.\lambda x.(\mu\delta.\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x)$.

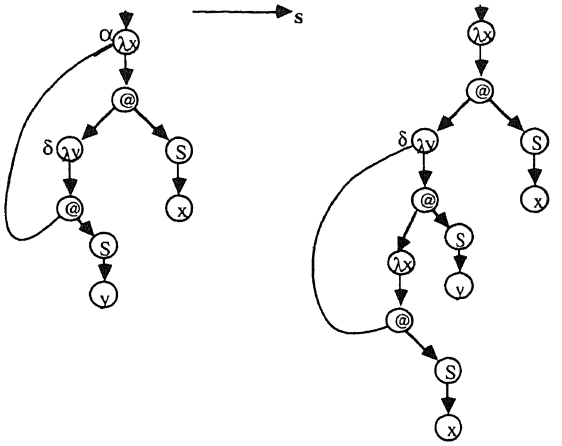


FIG. 20. Display of a substitution step.

of M. Moreover, in $\lambda\mu$ one step of unwinding is not enough to make the redex explicit. Another application of the μ-rule is necessary, this causes another copy of M to be made. This avoids the out of sync phenomenon.

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

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

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 after the soundness of $\lambda\Theta$ has been proved.

8. SOUNDNESS OF $\lambda\Theta$

In order to define the tree unwinding of a recursion system we first introduce the notion of expansion of a term. Let $M \xrightarrow[GK(\mathrm{as}^*)]{}\!\!\!\!\!\rightarrow^n N$ denote n-steps of the Gross–Knuth strategy applied to the acyclic substitution redexes occurring in the first equation of M (i.e., all acyclic substitution redexes in the first equation of M are

performed). If the first equation of M does not contain any acyclic substitution redexes we still write $M \xrightarrow[GK(as^*)]{} N$. For example,

$$\alpha = \mathsf{F}\gamma\gamma,\ \gamma = \mathsf{G}\gamma \xrightarrow[GK(as^*)]{}{}^{2} \alpha = \mathsf{F}(\mathsf{GG}\gamma)(\mathsf{GG}\gamma),\ \gamma = \mathsf{G}\gamma.$$

DEFINITION 8.1. Let g be $\alpha_1 = M_1, \ldots, \alpha_m = M_m$, and $\alpha = \alpha_1$, $g \xrightarrow[GK(as^*)]{}{}^{n} \alpha = M, g$. Then the nth expansion of g, written as $T^n(g)$, is the term $M[\alpha_1 := \Omega, \ldots, \alpha_m := \Omega]$.

Due to the monotonicity of expansion with respect to the $\leqslant$-ordering (i.e., the ordering axiomatized by $\Omega \leqslant t$, for any tree t), we define tree unwinding as follows.

DEFINITION 8.2. Given a recursion system g, the tree unwinding of g, written as $T(g)$, is $\lim_{n \to \infty} T^n(g)$.

Using the infinitary λ-calculus we can now formulate a soundness criterion for transformations of $\lambda\Theta$-expressions. (Also the various transformations in Section 9 satisfy this criterion.) We remind the reader that an Ω-step consists of the application of the following rule:

$$M \to \Omega \qquad \text{if } M \text{ has no weak head normal form.}$$

DEFINITION 8.3. Let g, g' be two recursion systems. We will say that a transformation $g \to g'$ is *sound* (with respect to the infinitary λ-calculus) if $T(g) \xrightarrow[\beta\Omega]{}{}^{\leqslant \omega} T(g')$. (Here $\xrightarrow[\beta\Omega]{}{}^{\leqslant \omega}$ denotes possibly infinitary reduction, that is a sequence of ω or less (possibly 0) β or Ω-steps.)

THEOREM 8.4. *The $\lambda\Theta$-calculus is sound with respect to the infinitary lambda calculus.*

Proof. We will prove the result for a single step, the result for multiple steps follows from the compression lemma of the infinitary lambda calculus. If $g \to_c g_1$, $g \to_n g_1$ or $g \to_s g_1$, then g and g_1 are bisimilar graphs and therefore $T(g) = T(g_1)$ (see [AK96]). If g rewrites to g_1 by reducing a β-redex, say ρ, then, since acyclic substitution commutes with β (Lemma 7.12) and the descendant of an as*-redex (i.e., an acyclic substitution redex occurring in the first equation) is still an as*-redex, the following holds,

$$g \to_\beta g_1 \Rightarrow \forall n,\ T^n(g) \twoheadrightarrow_\beta T^n(g_1),$$

where in the tree reduction all the descendants of ρ are reduced. Next, we will show that there exists a t such that $T(g) \xrightarrow[\beta\Omega]{}{}^{\leqslant \omega} t$ and $t = T(g_1)$. If $T(g)$ does not contain any descendants of ρ then we define $t = T(g)$; this may happen if the β-redex is garbage collected during the unwinding. If $T(g)$ contains an infinite number of descendants it means that the β-redex in g lies on a cycle. That is, g contains an equation of the form $\alpha = C[(\lambda x.P)\,Q]$, where either the context $C[\,\Box\,]$, P, or Q contains a reference to β, and g contains equations of the form

$\beta = C_1[\beta_1], ..., \beta_n = C_n[\alpha]$. In the following, without loss of generality, we let β be α. Let us assume that the context $C[\,\square\,]$ is empty and either P is α or P is x and Q is α. That is, g contains an equation of one of the following two forms: $\alpha = (\lambda x.x)\,\alpha$ or $\alpha = (\lambda x.\alpha)\,Q$. These redexes lead to the following rewritings:

$$\alpha = (\lambda x.x)\,\alpha \to_\beta \alpha = \alpha$$

and

$$\alpha = (\lambda x.\alpha)\,Q \to_\beta \alpha = \alpha.$$

In this case it is not true that $T(g) \xrightarrow[\beta]{}{}^{\leqslant\omega} T(g_1)$. In fact, $T(g)$ rewrites to itself only. The problem is that there always exists a redex at depth 0. However, the following holds:

$$T(g) = T(C[(\lambda x.P)\,Q]) \xrightarrow[\Omega]{} T(C[\Omega]) = T(C[P[x := Q]]) = T(g_1).$$

In the other cases, we can define t by doing a complete development of all the descendants of ρ that occur in $T(g)$. The next step is to prove that $t = T(g_1)$. We first show that $T^n(g_1) \leqslant t$. Let $T(g)$ rewrite to t' by doing all the β-redexes that are reduced in the reduction $T^n(g) \xrightarrow[\beta]{} T^n(g_1)$, then $T^n(g_1) \leqslant t'$ and $t' \twoheadrightarrow_\beta t$. Since all the descendants of ρ contained in t' correspond to an Ω in $T^n(g_1)$ we have $T^n(g_1) \leqslant t$. $t \leqslant T(g_1)$ since each finite approximation of t can be obtained by reducing a finite approximation of g. ∎

9. 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, and 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 $\lambda\mu$-calculus and of the $\lambda\sigma$-calculi [ACCL91, Cur93, Les94] with horizontal sharing and vertical sharing, respectively. 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

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

which corresponds to the letrec expression

$$\begin{aligned}&\mathsf{letrec}\ \alpha = \lambda x.\mathsf{letrec}\ \gamma = \lambda y.\mathsf{G}((\alpha y), \gamma)\\ &\qquad\qquad\qquad\qquad \mathsf{in}\ \mathsf{F}(\gamma x)\\ &\mathsf{in}\ \alpha.\end{aligned}$$

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].

After having presented the graphical representation of $\lambda\phi$ terms, we discuss the basic system $\lambda\phi_0$. $\lambda\phi_0$ is based on the confluent notion of acyclic substitution (applied also to the external part of a box); it does not contain rules for the manipulation of boxes except the empty ones. We show that $\lambda\phi_0$ is confluent and that the λ-calculus can be defined in it. We then present $\lambda\phi_1$, which is obtained by extending $\lambda\phi_0$ with some box distribution rules whose job is to move a box construct as far as possible down a term until a variable is reached. We show that the $\lambda\mu$-calculus is directly definable in $\lambda\phi_1$ and how to translate the $\lambda\sigma$-calculus into $\lambda\phi_1$. We prove confluence of $\lambda\phi_1$. Finally, we extend $\lambda\phi_1$ with rules to enlarge the scope of a box and to merge boxes when possible. The calculus so obtained, called $\lambda\phi_2$, is also shown to be confluent in the presence of orthogonal term or term graph rewriting system.

9.1. Graphical Representation of Modular λ-Graphs

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 9.1. (i) The terms

$$(a)\quad \langle \alpha \mid \alpha = F(\beta), \beta = G(\alpha)\rangle$$

$$(b)\quad \langle H(\alpha, \beta) \mid \alpha = F(\beta), \beta = G(\alpha)\rangle$$

$$(c)\quad \langle H(H(\alpha, \beta), \gamma)) \mid \alpha = H(\beta, \alpha)\rangle$$

are displayed in Fig. 21. Note that the free variables are drawn outside the box, as in Fig. 21c.

(ii) The terms

$$(a)\quad \langle H(\langle \alpha \mid \alpha = F(\alpha)\rangle, \beta) \mid \beta = F(\langle \gamma \mid \gamma = F(\delta), \delta = G(\gamma)\rangle)$$

$$(b)\quad \langle H(\alpha', \beta) \mid \alpha' = \langle \alpha \mid \alpha = F(\alpha)\rangle, \beta = F(\gamma'), \gamma' = \langle \gamma \mid \gamma = F(\delta), \delta = G(\gamma)\rangle\rangle$$

are shown in Fig. 22. Note the "external names" α', γ' of the boxes in Fig. 22b.

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

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

is shown in Fig. 23. Note that multiple references to a box are aiming straight at its leading node.

9.2. *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 $\langle t \mid E\rangle$. As in the previous section, "F orthogonal to a sequence of equations E and to a term t" means that the recursion variables of F do not intersect with the free variables of E and t. We denote this property by $F \perp E, t$. The recursion equation $\delta = \delta$ in the *black hole* rules stands for the sequence $\delta = \delta_1, \ldots, \delta_n = \delta$. 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

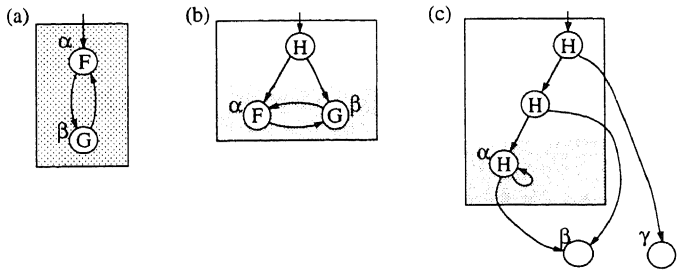


FIG. 21. Graphs associated to $\lambda\phi$-terms.

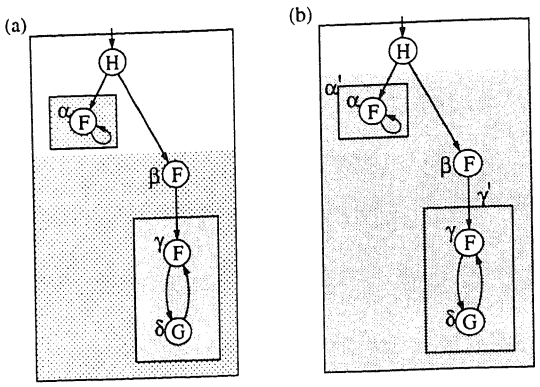


FIG. 22. Graphs associated to $\lambda\phi$-terms.

term matching the left-hand side of the rule. For example, α is greater than δ in the following underlined term,

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

even though g contains a cycle between α and δ. However, this cycle goes through γ, which is defined outside the internal box (see Fig. 24). The $\lambda\phi_0$-calculus is given next.

DEFINITION 9.2. The following clauses define the syntax and basic reduction axioms of the $\lambda\phi_0$-calculus.

SYNTAX:

$$t ::= \alpha \mid \mathsf{F}^n(t_1, ..., t_n) \mid \lambda\alpha.t \mid t_0 t_1 \mid \langle t_0 \mid \alpha_1 = t_1, ..., \alpha_n = t_n \rangle$$

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

In a term $\langle t_0 \mid \alpha_1 = t_1, ..., \alpha_n = t_n \rangle$ all the recursion variables α_i, $1 \leqslant i \leqslant n$, are distinct from each other.

REDUCTION AXIOMS:

β-rule:

$$(\lambda\alpha.t)\, s \qquad \to_\beta \ \langle t \mid \alpha = s \rangle$$

External substitution:

$$\langle C[\delta] \mid \delta = s, E \rangle \qquad \to_{\mathsf{es}} \ \langle C[s] \mid \delta = s, E \rangle$$

Acyclic substitution:

$$\langle t \mid \alpha = C[\delta], \delta = s, E \rangle \ \to_{\mathsf{as}} \ \langle t \mid \alpha = C[s], \delta = s, E \rangle \qquad \text{if} \quad \alpha > \delta$$

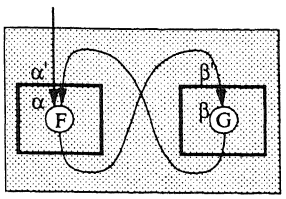


FIG. 23. Mutually dependent cyclic boxes.

Black hole:

$$\langle C[\delta] \mid \delta = \delta, E\rangle \quad \to_{\bullet} \quad \langle C[\bullet] \mid \delta = \delta, E\rangle$$

$$\langle t \mid \alpha = C[\delta], \delta = \delta, E\rangle \quad \to_{\bullet} \quad \langle t \mid \alpha = C[\bullet], \delta = \delta, E\rangle \quad \text{if} \quad \alpha > \delta$$

Garbage collection rules:

$$t^{E,F} \quad \to_{gc} \quad t^{E} \quad \text{if} \quad F \neq \varepsilon \quad \text{and} \quad F \perp E, t$$

$$\langle t \mid \rangle \quad \to_{gc} \quad t$$

Note that we have dropped the distinction between lambda bound variables and recursion variables. α-equivalent $\lambda\phi$-terms and terms that are obtained by a 1–1 renaming of recursion variables are identified. In the β-rule notice the role change of the bound variable, previously bound by λ, afterward bound by the recursion construct $\langle \mid \rangle$. The β-rule now becomes strongly normalizing. For example, $(\lambda\alpha.\alpha\alpha)(\lambda\alpha.\alpha\alpha) \to_{\beta} \langle \alpha\alpha \mid \alpha = \lambda\alpha.\alpha\alpha\rangle$, which does not contain any β-redex. In order to proceed with the computation external substitution has to be applied, yielding $\langle (\lambda\alpha.\alpha\alpha)\,\alpha \mid \alpha = \lambda\alpha.\alpha\alpha\rangle$. *External substitution* allows us to "extract" a tree-like prefix without duplicating the environment E. An external substitution redex corresponds to an as*-redex, introduced in Section 8. The cyclic binding $\delta = \delta$ in the black hole rule allows the reduction of $\langle \delta \mid \delta = \delta_1, \delta_1 = \delta\rangle$ to $\bullet$. This reduction would not have been possible if instead of $\delta = \delta$ we simply had $\delta = \delta$. In this case the only possible rewriting would have been the following:

$$\langle \delta \mid \delta = \delta_1, \delta_1 = \delta\rangle \to_{es} \langle \delta_1 \mid \delta = \delta_1, \delta_1 = \delta\rangle \to_{es} \langle \delta \mid \delta = \delta_1, \delta_1 = \delta\rangle \twoheadrightarrow_{es} \cdots$$

No reduction can occur inside the environment since δ and δ_1 lie on the same cycle. Moreover, we have included the proviso $\alpha > \delta$ in the black hole rule to guarantee its confluence. Without it, we would have the following scenario:

$$\begin{array}{ccc} \langle \delta \mid \delta = \delta_1, \delta_1 = \delta\rangle & \xrightarrow[\bullet]{} & \langle \delta \mid \delta = \bullet, \delta_1 = \delta\rangle \\ \Big\downarrow{\scriptstyle \bullet} & & \Big\downarrow{\scriptstyle as} \\ \langle \delta \mid \delta = \delta_1, \delta_1 = \bullet\rangle & \xrightarrow[as]{} & \langle \delta \mid \delta = \bullet, \delta_1 = \bullet\rangle \end{array}$$

The proviso "$F \neq \varepsilon$" of the first *garbage collection* rule guarantees its strong normalization. Without that proviso we would have $t^{E} \to_{gc} t^{E}$.

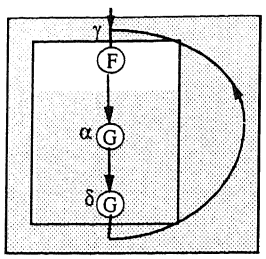
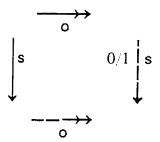


FIG. 24. Ordering among recursion variables.

THEOREM 9.3. *$\lambda\phi_0$ 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. By Lemma 7.5, s-reductions are confluent. Next, one s-step commutes with a sequence of o-steps (Notation: $\rightarrow^{0/1}$ 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 9.4. *The λ-calculus is directly definable in $\lambda\phi_0$.*

Proof. We have

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

The last step is justified by the fact that α cannot occur free in s. ∎

THEOREM 9.5. *Let R be an orthogonal term rewriting system. Then $\lambda\phi_0 \cup R$ is confluent.*

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

Rewriting with $\lambda\phi_0 \cup R$ is already quite interesting from the point of view of term graph rewriting, as it can handle horizontal (as shown in the following example) and vertical sharing.

EXAMPLE 9.6. Let CL be combinatory logic, with the rules

$$\mathsf{S}Z_1Z_2Z_3 \rightarrow Z_1Z_3(Z_2Z_3)$$

$$\mathsf{K}Z_1Z_2 \rightarrow Z_1$$

$$\mathsf{I}Z \rightarrow Z.$$

Then we have the following reduction in $\lambda\phi_0 \cup R$ (see also Fig. 25, where the lines dividing the graphs correspond to the division in external and internal part. Only the nodes reachable from the root are displayed):

$$\begin{aligned}
&\langle \alpha \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle && \to_{es} \\
&\langle \beta\beta \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle && \to_{es} \\
&\langle \mathsf{S}\gamma\gamma\beta \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle && \to_{CL} \\
&\langle \gamma\beta(\gamma\beta) \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle && \twoheadrightarrow_{es} \\
&\langle \mathsf{I}\beta(\mathsf{I}\beta) \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle && \twoheadrightarrow_{CL} \\
&\langle \beta\beta \mid \alpha = \beta\beta,\ \beta = \mathsf{S}\gamma\gamma,\ \gamma = \mathsf{I} \rangle.
\end{aligned}$$

Remark 9.7. As pointed out in Section 7, non-confluence is caused by a notion of cyclic substitution. This cyclic substitution is now absent in $\lambda\phi_0$. Thus, all counterexamples to confluence disappear in $\lambda\phi_0$ and in our subsequent extensions. This restriction does not limit the expressive power of our calculi with respect to execution. That is, they are powerful enough to simulate finite β-reductions in the infinitary calculus.

(i) Consider the system

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

which caused the first counterexample to confluence (see Section 4). In $\lambda\phi_0$ there is no way of making the implicit β-redexes $\alpha(\mathsf{S}y)$ and $\gamma(\mathsf{S}x)$ explicit by applying substitution inside the environment. Thus, in $\lambda\phi_0$ g does not rewrite to

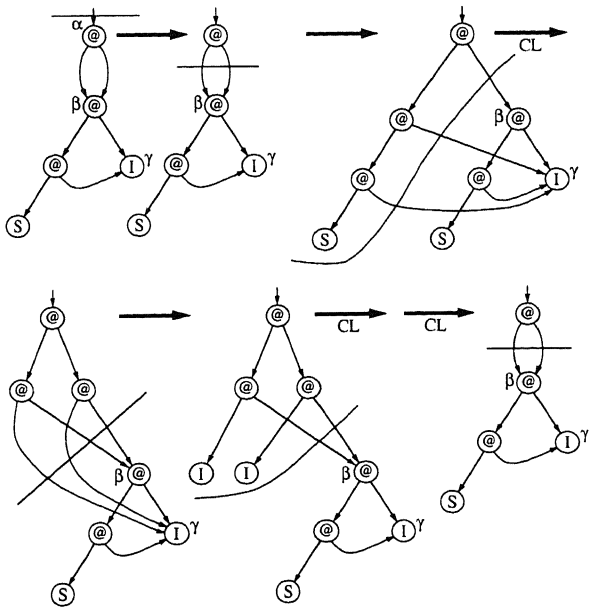


FIG. 25. Reduction in $\lambda\phi_0 \cup R$.

$\langle \alpha \mid \alpha = \lambda x.\alpha(\mathsf{S}^2 x)\rangle$. In $\lambda\phi_0$ we have the following reductions ($\twoheadrightarrow_{\beta\mathsf{esgc}}$ stands for $\rightarrow_\beta \rightarrow_{\mathsf{es}} \rightarrow_{\mathsf{gc}}$):

$$\begin{aligned}
&\langle \alpha \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_{\mathsf{es}}\\
&\langle \lambda x.\gamma(\mathsf{S}x) \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_{\mathsf{es}}\\
&\langle \lambda x.(\lambda y.\alpha(\mathsf{S}y))(\mathsf{S}x) \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \twoheadrightarrow_{\beta\mathsf{esgc}}\\
&\langle \lambda x.\alpha(\mathsf{S}^2 x) \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \rightarrow_{\mathsf{es}}\\
&\langle \lambda x.(\lambda x.\gamma(\mathsf{S}x))(\mathsf{S}^2 x) \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \twoheadrightarrow_{\beta\mathsf{esgc}}\\
&\langle \lambda x.\gamma(\mathsf{S}^3 x) \mid \alpha = \lambda x.\gamma(\mathsf{S}x),\ \gamma = \lambda y.\alpha(\mathsf{S}y)\rangle && \twoheadrightarrow\\
&\ldots
\end{aligned}$$

Note that independently of how many rewriting steps are performed, the information contained in g is $\lambda x.\Omega$, which is the infinite normal form of $T(g)$.

(ii) Consider the second counterexample (see Section 6):

$$\begin{aligned}
&g \equiv \langle \alpha \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\ \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y)\rangle && \rightarrow_{\mathsf{es}}\\
&\langle \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x) \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\ \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y)\rangle && \rightarrow_{\mathsf{es}}\\
&\langle \lambda x.\mathsf{F}((\lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y))(\mathsf{S}x), \mathsf{S}x) \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\\
&\quad \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y)\rangle && \twoheadrightarrow_{\beta\mathsf{esgc}}\\
&\langle \lambda x.\mathsf{F}(\mathsf{G}(\alpha(\mathsf{S}^2 x), \mathsf{S}^2 x), \mathsf{S}x) \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\ \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y)\rangle && \twoheadrightarrow\\
&\langle \lambda x.\mathsf{F}(\mathsf{G}(\mathsf{F}(\gamma(\mathsf{S}^3 x), \mathsf{S}^3 x), \mathsf{S}^2 x), \mathsf{S}x) \mid \alpha = \lambda x.\mathsf{F}(\gamma(\mathsf{S}x), \mathsf{S}x),\\
&\quad \gamma = \lambda y.\mathsf{G}(\alpha(\mathsf{S}y), \mathsf{S}y)\rangle && \twoheadrightarrow\\
&\ldots
\end{aligned}$$

Note that even though g cannot rewrite to a g_1 such that $T(g_1)$ is the tree on the right-hand side of Fig. 14, reductions in $\lambda\phi_0$ produce all finite approximations of that tree; e.g., the above reduction leads to the approximation $\lambda x.\mathsf{F}(\mathsf{G}(\mathsf{F}(\Omega, \mathsf{S}^3 x), \mathsf{S}^2 x), \mathsf{S}x)$.

9.3. *$\lambda\mu$ with Horizontal Sharing and $\lambda\sigma$ with Vertical Sharing*

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

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

However, the $\lambda\mu$-calculus is not directly definable in $\lambda\phi_0$; e.g.,

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

but

$$\begin{aligned}\phi[\![t]\!] &\equiv \langle \alpha \mid \alpha = \mathsf{F}(\alpha, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta)\rangle)\rangle \\ &\not\to\!\!\!\to_{\lambda\phi_0} \langle \alpha \mid \alpha = \mathsf{F}(\alpha, \mathsf{G}(\alpha, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta)\rangle))\rangle \equiv \phi[\![s]\!].\end{aligned}$$

To that end, we extend $\lambda\phi_0$ with the distribution rules of Table 1, 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: if F is $\alpha_1 = t_1, \ldots, \alpha_n = t_n$ then F^E stands for $\alpha_1 = t_1^E, \ldots, \alpha_n = t_n^E$; α bound by F means that F contains an equation of the form $\alpha = t$.

EXAMPLE 9.8. The reduction

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

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

$$\begin{aligned}&\langle \alpha \mid \alpha = \mathsf{F}(\alpha, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta)\rangle)\rangle && \to_{\mathsf{es}} \\ &\langle \alpha \mid \alpha = \mathsf{F}(\alpha, \langle \mathsf{G}(\alpha, \beta) \mid \beta = \mathsf{G}(\alpha, \beta)\rangle)\rangle && \to_{\mathsf{dF}} \\ &\langle \alpha \mid \alpha = \mathsf{F}(\alpha, \mathsf{G}(\langle \alpha \mid \beta = \mathsf{G}(\alpha, \beta)\rangle, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta)\rangle))\rangle && \to\!\!\!\to_{\mathsf{gc}} \\ &\langle \alpha \mid \alpha = \mathsf{F}(\alpha, \mathsf{G}(\alpha, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta)\rangle))\rangle.\end{aligned}$$

(See Fig. 26.)

In order to prove that $\lambda\mu$ is definable in $\lambda\phi_1$ we need some properties of the distribution and garbage collection rules, i.e., strong normalization and confluence. Using these properties we then show that the distribution and garbage collection rules unfold the system by pushing the box constructs next to the variables. Notation: $\to\!\!\!\to_{\mathsf{dgc}}$ is the reduction relation induced by the distribution and garbage collection rules.

LEMMA 9.9. $\to\!\!\!\to_{\mathsf{dgc}}$ *is strongly normalizing.*

TABLE 1

Distribution Rules

$(\lambda\alpha.t)^E$	$\to_{\mathsf{d}\lambda}$	$\lambda\alpha.t^E$	
$\mathsf{F}^n(t_1, \ldots, t_n)^E$	$\to_{\mathsf{dF}}$	$\mathsf{F}^n(t_1^E, \ldots, t_n^E)$	if $n \geqslant 1$
$(ts)^E$	$\to_{\mathsf{d}@}$	$t^E s^E$	
$\langle \alpha \mid F\rangle^E$	$\to_{\mathsf{d}\Box}$	$\langle \alpha \mid F^E\rangle$	if α is bound by F

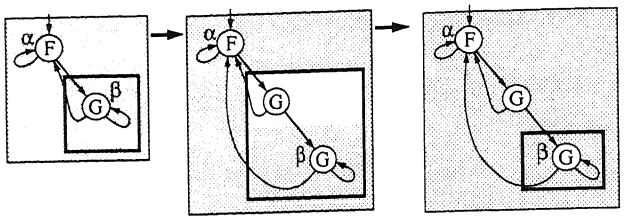


FIG. 26. Analysis of μ-step.

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:

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

(We assume $\max\{\ \}$ 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

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

then

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

(ii) If

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

then

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

We first restrict our attention to the distribution rules only (written as $\to_{\mathsf{d}}$). We show the following fact:

Fact. $C[R] \to_{\mathsf{d}} C[R_1] \Rightarrow d(C[R]) = d(C[R_1])$.
By induction on the structure of $C[\,\Box\,]$.

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

— $d((\lambda\alpha.t)^E) = 1 + d(\lambda\alpha.t) + d(E)$
$= 1 + (1 + d(t) + d(E))$
$= 1 + d(t^E)$
$= d(\lambda\alpha.t^E)$.

— $d((st)^E) = 1 + d(st) + d(E)$
$= 1 + 1 + \max\{d(s), d(t)\} + d(E)$
$= 1 + \max\{1 + d(s), 1 + d(t)\} + d(E)$
$= 1 + \max\{1 + d(s) + d(E), 1 + d(t) + d(E)\}$
$= 1 + \max\{d(s^E), d(t^E)\} = d(s^E t^E)$.

— $d(\mathsf{F}^n(t_1, \ldots, t_n)^E) = 1 + d(\mathsf{F}^n(t_1, \ldots, t_n)) + d(E)$
$= 1 + 1 + \max\{d(t_1), \ldots, d(t_n)\} + d(E)$
since $n \geqslant 1$ $= 1 + \max\{1 + d(E) + d(t_1), \ldots, 1 + d(E) + d(t_n)\}$
$= 1 + \max\{d(t_1^E), \ldots, d(t_n^E)\}$
$= d(\mathsf{F}^n(t_1^E, \ldots, t_n^E))$.

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

— $d(\langle\langle \alpha \mid \alpha_1 = t_1, ..., \alpha_n = t_n\rangle \mid E\rangle)$

$= 1 + d(\langle \alpha \mid \alpha_1 = t_1, ..., \alpha_n = t_n\rangle) + d(E)$

$= 1 + 1 + \max\{d(t_1), ..., d(t_n)\} + d(E)$

since $n \geqslant 1$

$= 1 + \max\{1 + d(t_1) + d(E), ..., 1 + d(t_n) + d(E)\}$

$= 1 + \max\{d(t_1^E), ..., d(t_n^E)\}$

$= d(\langle \alpha \mid \alpha_1 = t_1^E, ..., \alpha_n = t_n^E\rangle)$.

The proviso of the $\rightarrow_{d\Box}$-rule guarantees that n is greater than zero.

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

$d(C_1[R]\ t) = 1 + \max\{d(C_1[R]), d(t)\}$ Induction hypothesis

$= 1 + \max d(C_1[R_1]), d(t)\}$

$= d(C_1[R_1]\ t)$.

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

We are now ready to show that

$$g \equiv C[R] \rightarrow_d C[R_1] \equiv g_1 \Rightarrow |g| > |g_1|.$$

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

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

— $\langle \lambda\alpha.t \mid E\rangle \rightarrow_{d\lambda} \lambda\alpha.\langle t \mid E\rangle$. 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.

— $\langle st \mid E\rangle \rightarrow_{d@} \langle s \mid E\rangle\langle t \mid E\rangle$. 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.

— $\langle \mathsf{F}(t_1, ..., t_n) \mid E\rangle \rightarrow_{d\mathsf{F}} \mathsf{F}(\langle t_1 \mid E\rangle, ..., \langle t_n \mid E\rangle)$. Same as the case above.

— $\langle\langle \alpha \mid \alpha_1 = t_1, ..., \alpha_n = t_n\rangle \mid E\rangle \rightarrow_{d\Box} \langle \alpha \mid \alpha_1 = \langle t_1 \mid E\rangle, ..., \alpha_n = \langle t_n \mid E\rangle\rangle$. The index of the outside box is $1 + \max\{d(t_1), ..., d(t_n)\}$ 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 9.10. If we change the current distribution rule over a box construct to

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

then the distribution rules will no longer be strongly normalizing; e.g.,

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

LEMMA 9.11. $\twoheadrightarrow_{\mathsf{dgc}}$ *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: $t[\alpha_1 := \langle \alpha_1 \mid E \rangle, \ldots, \alpha_n := \langle \alpha_n \mid E \rangle]$ denotes a simultaneous substitution. $\mathsf{nf}_{\mathsf{dgc}}(t)$ is the normal form with respect to the distribution and garbage collection rules.

LEMMA 9.12 (Unfolding Lemma). *Let t be a term and E be $\alpha_1 = s_1, \ldots, \alpha_n = s_n$. Then*

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

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:

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

— t is $t_1 t_2$. We have

$$\begin{aligned}
&\langle t_1 t_2 \mid \alpha_1 = s_1 \rangle && \to_{\mathsf{d}(a)} \\
&\langle t_1 \mid \alpha_1 = s_1 \rangle \langle t_2 \mid \alpha_1 = s_1 \rangle && \twoheadrightarrow_{\mathsf{dgc}} \text{ Induction hypothesis} \\
&t_1[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1 \rangle]\, t_2[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1 \rangle] \equiv \\
&(t_1 t_2)[\alpha_1 := \langle \alpha_1 \mid \alpha_1 = s_1 \rangle].
\end{aligned}$$

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

— t is $\lambda\alpha.t_1$. We have

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

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

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

— t is not a normal form. Then

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

By the induction hypothesis,

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

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

THEOREM 9.13. *$\lambda\mu$ is directly definable in $\lambda\phi_1$.*

Proof. We show that

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

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

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

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

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:

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

Otherwise:

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

— t is $t_1 t_2$.

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

— t is $\mathsf{F}^n(t_1, \ldots, t_n)$. Same as the case above.

— t is $\langle\alpha \mid E\rangle$. Without loss of generality, let us assume E to be $\alpha = s$.

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

— t is not a normal form. It follows immediately from the induction hypothesis. ∎

PROPOSITION 9.15. *Let s be a term and E a sequence of equations. Then*

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

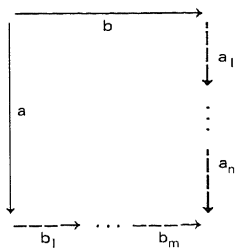
Proof. By structural induction on $C[\Box]$ and Proposition 9.14. ∎

Intermezzo 9.16. In the proof of confluence of $\lambda\phi_1$ we will use the decreasing diagram method proposed by van Oostrom [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 9.17. Let $|.|$ be a measure from strings of labels to multisets of labels. If $a_1, \ldots, a_n$ are labels,

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

Then the diagram



is decreasing if $\{\{a, b\}\} \geqslant |ab_1 \cdots b_m|$ and $\{\{a, b\}\} \geqslant |ba_1 \cdots a_n|$.

THEOREM 9.18. *If a labeled 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 9.19. *$\lambda\phi_1$ is confluent.*

Proof. We call the external and acyclic substitution reductions 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:

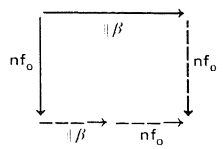
$t \to_{\mathsf{nf_o}} s$: if s is the normal form of t with respect to the o-rules;

$t \to_{\|\mathsf{s}} s$: if s is obtained from t by a complete development of a set, possibly empty, of substitution redexes (this is possible since s-substitutions are confluent by developments, see Theorem 7.5);

$t \to_{\|\beta} s$: 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, $t \to_{\|\beta} s$ if and only if $t \twoheadrightarrow_\beta s$.

Let us first prove the weak confluence diagrams.

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



(9.4)

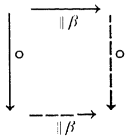
Let us first point out that the only obstacle to β commuting with o-reductions is caused by the distribution of an environment over an application:

$$((\lambda\alpha.s)\,t)^E \to_{\mathsf{d}\alpha} (\lambda\alpha.s)^E\, t^E.$$

The right-hand side of the reduction is no longer a β-redex. We call this distribution-step an interfering $\mathsf{d}\alpha^\star$-reduction. The distribution over lambda that restores the β-redex is denoted by $\mathsf{d}\lambda^\star$:

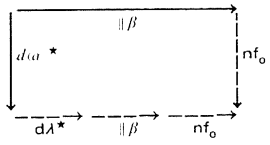
$$((\lambda\alpha.s)\,t)^E \to_{\mathsf{d}\alpha^\star} (\lambda\alpha.s)^E\, t^E \to_{\mathsf{d}\lambda^\star} (\lambda\alpha.s^E)\,t^E.$$

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



(9.5)

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

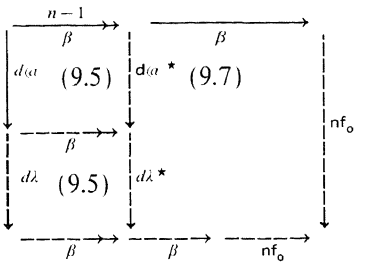


(9.6)

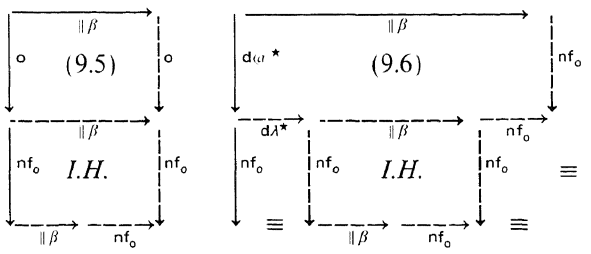
For a single β-step:

$$\begin{array}{ccc} ((\lambda\alpha.t)s)^E & \xrightarrow[\beta]{\qquad\qquad} & \langle t \mid \alpha = s\rangle^E \\ {\scriptstyle \mathsf{d}\alpha^\star}\downarrow & & = {\scriptstyle \mathsf{dgc}} \\ (\lambda\alpha.t)^E\, s^E & \xrightarrow[\mathsf{d}\lambda^\star]{} (\lambda\alpha.t^E)\, s^E & \xrightarrow[\beta]{} \langle t^E \mid \alpha = s^E\rangle \end{array} \tag{9.7}$$

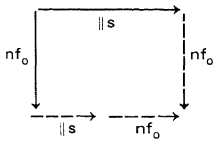
($\langle t \mid \alpha = s \rangle^E =_{\mathsf{dgc}} \langle t^E \mid \alpha = s^E \rangle$ follows from Proposition 9.14.) Since $\twoheadrightarrow_{\mathsf{dgc}}$ is confluent, $\langle t \mid \alpha = s \rangle^E$ and $\langle t^E \mid \alpha = s^E \rangle$ have the same normal form. For the number of β-steps greater than one we first re-order the β-reduction so that the interfering step is the last step. We then have:



We are now ready to prove our result (i.e., diagram (9.4)). 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:



— s-reductions and o-reductions. The goal is to prove the following commuting diagram:



(9.8)

We remind the reader that the bottom $\rightarrow_{\|s}$-reduction of the above diagram might correspond to an empty reduction; e.g.,

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

The bottom $\rightarrow_{nf_o}$-step of diagram (9.8) is due to the interference between external substitution and the distribution of an environment over a box construct:

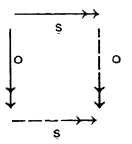
$$\begin{array}{ccc} \langle\langle \alpha \mid \alpha = s\rangle \mid F\rangle & \xrightarrow[\text{es}]{} & \langle\langle s \mid \alpha = s\rangle \mid F\rangle \\ {\scriptstyle d\Box}\downarrow & & = {\scriptstyle dgc} \\ \langle \alpha \mid \alpha = \langle s \mid F\rangle\rangle & \xrightarrow[\text{es}]{}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

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:

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

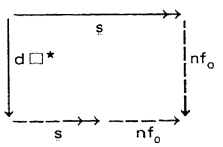
Thus, the distribution of an environment E over a box construct of the form $\langle \alpha \mid F\rangle$ is interfering if $\langle \alpha \mid F\rangle$ is either an es or an as-redex.

By associating to each variable α a weight, say n, as in the proof of strong normalization of $\twoheadrightarrow_{\underline{as}\beta}$ and to a variable α^n a depth of n instead of 0, we can show, following the steps of the proof of Lemma 9.9, that 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 the case of non-interference, $\twoheadrightarrow_s$ commutes with $\twoheadrightarrow_o$:

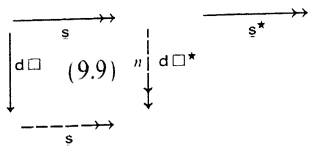


(9.9)

If the o-reduction interferes with s-reductions, we prove



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



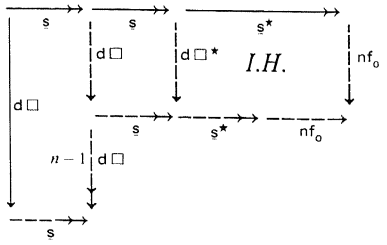
Note that the n $d\Box^*$-redexes are disjoint from each other; that is, the corresponding boxes are not contained within each other. We show by induction on n that we can close the above diagram.

— $n = 1$: 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

$$
\begin{array}{lll}
\langle \underline{\alpha_1} \mid \alpha_1 = C_1[\underline{\alpha_2}], & \xrightarrow[\underline{s}^*]{} & \langle C_1[C_2[s]] \mid \alpha_1 = C_1[C_2[s]], \\
\quad \alpha_2 = C_2[\underline{\alpha_3}], & & \quad \alpha_2 = C_2[s], \\
\quad \alpha_3 = s\rangle^E & & \quad \alpha_3 = s\rangle^E \\
{\scriptstyle d\Box^*}\downarrow & & \quad = {\scriptstyle dgc} \\
\langle \underline{\alpha_1} \mid \alpha_1 = C_1[\underline{\alpha_2}]^E, & \xrightarrow[\underline{s}]{} & \langle C_1[C_2[s^E]^E]^E \mid \alpha_1 = C_1[C_2[s^E]^E]^E, \\
\quad \alpha_2 = C_2[\underline{\alpha_3}]^E, & & \quad \alpha_2 = C_2[s^E]^E, \\
\quad \alpha_3 = s^E\rangle & & \quad \alpha_3 = s^E\rangle. \\
\langle C_1[C_2[s]] \mid \alpha_1 = C_1[C_2[s]], & = {\scriptstyle dgc} & \langle C_1[C_2[s^E]^E]^E \mid \alpha_1 = C_1[C_2[s^E]^E]^E, \\
\quad \alpha_2 = C_2[s], & & \quad \alpha_2 = C_2[s^E]^E, \\
\quad \alpha_3 = s\rangle^E & & \quad \alpha_3 = s^E\rangle
\end{array}
$$

which follows from Proposition 9.15.

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



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

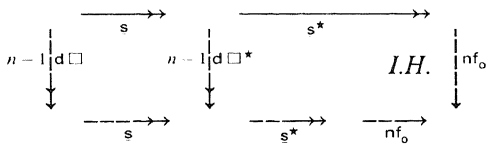
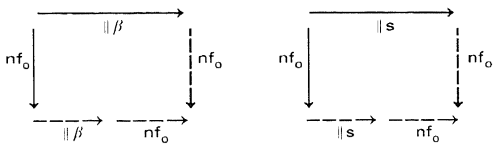
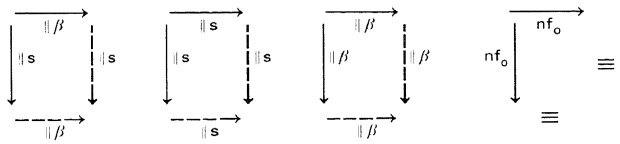


Diagram (9.8) follows by noetherian induction with respect to $\underline{s} \cup o$-reductions.

Summarizing, we have proved the following commuting diagrams:



Moreover, we also know the following diagrams:



According to the ordering $\|\beta > \mathsf{nf}_o < \|\mathsf{s}$, the above diagrams are decreasing, and thus by Theorem 9.18 $\lambda\phi'_1$ 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,

$$t \to_{\lambda\phi_1} t \Rightarrow \exists s,\ t \twoheadrightarrow_{\lambda\phi'_1} s \text{ and } t_1 \twoheadrightarrow_{\lambda\phi'_1} s.$$

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

$$t \twoheadrightarrow_{\lambda\phi'_1} t_1 \Rightarrow t \twoheadrightarrow_{\lambda\phi_1} t_1. \quad \blacksquare$$

INTERMEZZO 9.20. $\lambda\phi_1$ 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:

$$\mathscr{T}[\![x]\!] = x$$

$$\mathscr{T}[\![ab]\!] = \mathscr{T}[\![a]\!]\,\mathscr{T}[\![b]\!]$$

$$\mathscr{T}[\![\lambda x.a]\!] = \lambda x.\mathscr{T}[\![a]\!]$$

$$\mathscr{T}[\![a[s]]\!] = \langle\langle \mathscr{T}[\![a]\!] \mid \mathscr{S}\mathrm{in}_{\{\}}[\![s]\!]\rangle \mid \mathscr{S}\mathrm{out}_{\{\}}[\![s]\!]\rangle$$

$$\mathscr{S}\text{in}_{\text{var}}[\![id]\!] = \varepsilon$$

$$\mathscr{S}\text{in}_{\text{var}}[\![(a/x).s]\!] = \begin{cases} \mathscr{S}\text{in}_{\text{var}}[\![s]\!] & x \in \text{var} \\ x = x',\ \mathscr{S}\text{in}_{\text{var} \cup \{x\}}[\![s]\!] & x \notin \text{var} \end{cases}$$

$$\mathscr{S}\text{out}_{\text{var}}[\![id]\!] = \varepsilon$$

$$\mathscr{S}\text{out}_{\text{var}}[\![(a/x).s]\!] = \begin{cases} \mathscr{S}\text{out}_{\text{var}}[\![s]\!] & x \in \text{var} \\ x' = \mathscr{T}[\![a]\!],\ \mathscr{S}\text{out}_{\text{var} \cup \{x\}}[\![s]\!] & x \notin \text{var}. \end{cases}$$

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

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

is translated as

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

The binding $x = x'$ is generated by $\mathscr{S}\text{in}_{\text{var}}$ and the binding $x' = \text{cons } 1\ x$ is generated by $\mathscr{S}\text{out}_{\text{var}}$.

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:

Var_1:

$$x[(a/x).s] = a$$

Var_2:

$$x[(a/y).s] = x[s] \quad \text{if} \quad x \neq y$$

Var_3:

$$x[id] = x.$$

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

Abs:

$$(\lambda x.a)[s] = \lambda y.a[(y/x).s] \quad \text{if } y \text{ occurs in neither } a \text{ nor } s$$

App:

$$(ab)[s] = (a[s])(b[s]).$$

9.4. A Calculus for Modular Lambda Graph Rewriting

Until now we have kept the internal structure of a term. For example, we distinguish between the following two terms t_1 and t_2, respectively:

$$\langle \alpha \mid \alpha = \underline{\langle \delta * \delta \mid \delta = t \rangle} \rangle \qquad \langle \alpha \mid \alpha = \delta * \delta, \delta = t \rangle.$$

However, we would like to consider the underlined box in t_1 as syntactic noise. To that end, we rewrite t_1 to t_2 by applying the following box elimination rule:

$$\langle t \mid \alpha = s^E, F\rangle \to_{\square_1} \langle t \mid \alpha = s, E, F\rangle. \tag{9.10}$$

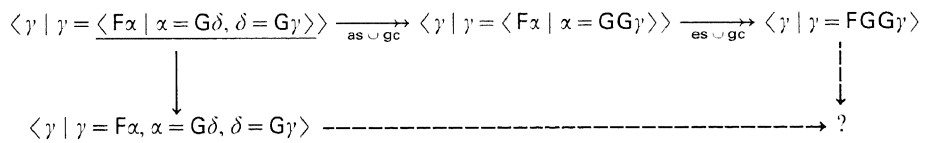
The application of this rule becomes at times necessary in order to capture the amount of sharing in lazy implementations of functional languages, as described by Ariola *et al.* [AFM+95, AF]. Consider the following reduction:

$$\langle (\alpha 3) + (\alpha 4) \mid \alpha = \langle \lambda\gamma.\gamma * \delta \mid \delta = 1 + 1\rangle\rangle \to\!\!\to \langle 3 * \delta \mid \delta = 1 + 1\rangle + \langle 4 * \delta \mid \delta = 1 + 1\rangle.$$

An unnecessary copy of the redex $1 + 1$ has been performed; the reduction of this redex can be shared between the two different applications of α. This sharing occurs if before substituting for α, the box surrounding the lambda is eliminated, as described below:

$$\begin{aligned}\langle (\alpha 3) + (\alpha 4) \mid \alpha = \langle \lambda\gamma.\gamma * \delta \mid \delta = 1 + 1\rangle\rangle &\to_{\square_2} \langle (\alpha 3) + (\alpha 4) \mid \alpha = \lambda\gamma.\gamma * \delta, \delta = 1 + 1\rangle \\ &\to\!\!\to \langle (3 * \delta) + (\alpha 4) \mid \alpha = \lambda\gamma.\gamma * \delta, \delta = 1 + 1\rangle \\ &\to \langle (3 * \delta) + (\alpha 4) \mid \alpha = \lambda\gamma.\gamma * \delta, \delta = 2\rangle.\end{aligned}$$

However, not all the boxes can be eliminated. Consider the following example:



We have removed the underlined box which, as depicted in Fig. 24, is on a cycle. Once this cyclic box is removed the substitutions for α and δ will no longer be acyclic substitutions. This means that we need to distinguish between two kind of boxes: *acyclic* and *cyclic*. The boxes of Fig. 26 that are drawn with heavy lines are examples of cyclic boxes. The boxes of Fig. 27 are acyclic, since we require the cyclic path to go through the internal part of the box and be within the *parent box*. (A parent box of a box is the smallest box properly containing it.) Only acyclic boxes can be removed safely. Note that boxes of the form $\langle t \mid \rangle$ can always be safely removed. Also, the underlined box in the term:

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

cannot be removed since γ will get out of scope. 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 [AL94].

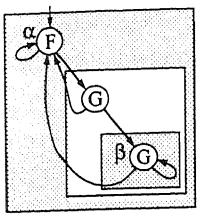


FIG. 27. Graph of $\langle \alpha \mid \alpha = \mathsf{F}(\alpha, \langle \mathsf{G}(\alpha, \langle \beta \mid \beta = \mathsf{G}(\alpha, \beta) \rangle) \mid \rangle) \rangle$.

Following the above discussion we add a proviso to rule (9.10), obtaining

$$\langle t \mid \alpha = s^E, F \rangle \to_{\square\square} \langle t \mid \alpha = s, E, F \rangle \qquad \text{if} \quad s^E \text{ is acyclic.}$$

We also merge external boxes with the rule

$$(t^E)^F \to_{\square\mathsf{m}} t^{E,F}.$$

However, we still run into problems if in the following example s^E is a cyclic box:

$$\begin{array}{ccc} \langle t \mid \alpha = (s^E)^F \rangle & \xrightarrow[\square\mathsf{m}]{} & \langle t \mid \alpha = s^{E,F} \rangle \\ \Big\downarrow{\scriptstyle \square\square} & & \Big\downarrow \\ \langle t \mid \alpha = s^E, F \rangle & \dashrightarrow & ? \end{array} \tag{9.11}$$

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

$$\langle t \mid \alpha = s^{E_1, E}, F \rangle \to_{\square\square} \langle t \mid \alpha = s^{E_1}, E, F \rangle \qquad \text{if} \quad E \neq \varepsilon \text{ and } E_1, \alpha > E,$$

where $\alpha > E$ means that α and the 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 the new rule we can now close diagram (9.11):

$$\begin{array}{ccc} \langle t \mid \alpha = (s^E)^F \rangle & \xrightarrow[\square\mathsf{m}]{} & \langle t \mid \alpha = s^{E,F} \rangle \\ \Big\downarrow{\scriptstyle \square\square} & & \Big\downarrow{\scriptstyle \square\square} \\ \langle t \mid \alpha = \langle s^E \mid \rangle, F \rangle & \xrightarrow[\mathsf{gc}]{} & \langle t \mid \alpha = s^E, F \rangle. \end{array}$$

We need to move equations out of a lambda to cope with the following diagram:

$$\begin{array}{ccc} \langle t \mid \alpha = \langle \lambda\gamma.s^E \rangle^{E_1} \rangle & \xrightarrow[\square\mathsf{m}]{} & \langle t \mid \alpha = \lambda\gamma.s^E, E_1 \rangle \\ \Big\downarrow & & \\ \langle t \mid \alpha = \lambda\gamma.s^{E,E_1} \rangle & & \end{array}$$

The full set of rules is displayed in Table 2. The proviso $E \neq \varepsilon$ 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; e.g.,

$$\begin{array}{ccc} \langle\langle \alpha_1 \mid \alpha_1 = s_1, \alpha_2 = s_2 \rangle \mid F \rangle & \xrightarrow[\mathrm{d}\square]{} & \langle \alpha_1 \mid \alpha_1 = s_1^F, \alpha_2 = s_2^F \rangle \\ \Big\downarrow \square_{\mathrm{m}} & & \vdots \\ \langle \alpha_1 \mid \alpha_1 = s_1, \alpha_2 = s_2, F \rangle & \xrightarrow[\mathrm{c}]{} & \langle \alpha_1 \mid \alpha_1 = s_1, \alpha_2 = s_2', F, F' \rangle \end{array}$$

where s_2' and F' denote a renamed version of s_2 and F, respectively. The dashed vertical reductions consist of a sequence of $\square_{\square}$ steps followed by empty box removals. The new system is called $\lambda\phi_2$. We present all the rules for $\lambda\phi_0$, $\lambda\phi_1$, and $\lambda\phi_2$ in Table 3.

PROPOSITION 9.21. *Box elimination rules, garbage collection, and black hole rules are strongly normalizing.*

Proof. To each term t we associate a measure, written as $w(t)$, that consists of a multiset counting the distance to the root for every box and every equation. $w(t)$ is defined as follows:

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

inc adds one to each element of the multiset, i.e., $\mathrm{inc}(\{\!\{n_1, \ldots, n_m\}\!\}) = \{\!\{n_1 + 1, \ldots, n_m + 1\}\!\}$. For example, $w(\langle\langle \alpha \mid \alpha = \mathsf{F}\beta \rangle \mid \beta = \langle \delta \mid \delta = \mathsf{G}\delta \rangle\rangle) = \{\!\{0, 0, 0, 0, 1, 1\}\!\}$. It 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. ∎

TABLE II

Box Elimination Rules

$\lambda\alpha.t^{E_1 \cdot E}$	$\to_{\square\lambda}\ (\lambda\alpha.t^{E_1})^E$	if $E \neq \varepsilon$ and $E_1, \alpha > E$
$\mathsf{F}(t_1, \ldots, t_i^E, \ldots, t_n)$	$\to_{\square\mathsf{F}}\ \mathsf{F}(t_1, \ldots, t_i, \ldots, t_n)^E$	
$t^E s$	$\to_{\square@_1}\ (ts)^E$	
ts^E	$\to_{\square@_r}\ (ts)^E$	
$(t^E)^F$	$\to_{\square\mathrm{m}}\ t^{E,F}$	
$\langle t \mid \alpha = s^{E_1 \cdot E}, F \rangle$	$\to_{\square_{\square}}\ \langle t \mid \alpha = s^{E_1}, E, F \rangle$	if $E \neq \varepsilon$ and $E_1, \alpha > E$

TABLE III

Reduction Rules of $\lambda\phi_0$, $\lambda\phi_1$, and $\lambda\phi_2$

β-rule:			
$(\lambda\alpha.t)s$	$\to_{\beta}$	$\langle t \mid \alpha = s\rangle$	
External substitution:			
$\langle C[\delta] \mid \delta = s, E\rangle$	$\to_{es}$	$\langle C[s] \mid \delta = s, E\rangle$	
Acyclic substitution:			
$\langle t \mid \alpha = C[\delta], \delta = s, E\rangle$	$\to_{as}$	$\langle t \mid \alpha = C[s], \delta = s, E\rangle$	if $\alpha > \delta$
Black hole:			
$\langle C[\delta] \mid \delta =_0 \delta, E\rangle$	$\to_{\cdot}$	$\langle C[\bullet] \mid \delta =_0 \delta, E\rangle$	
$\langle t \mid \alpha = C[\delta], \delta =_0 \delta, E\rangle$	$\to_{\cdot}$	$\langle t \mid \alpha = C[\bullet], \delta =_0 \delta, E\rangle$	if $\alpha > \delta$
Garbage collection rules:			
$t^{E_1 \cdot E}$	$\to_{gc}$	t^E	if $F \neq \varepsilon$ and $F \perp E, t$
$\langle t \mid \rangle$	$\to_{gc}$	t	
$\lambda\phi_0$			
Distribution rules:			
$(\lambda\alpha.t)^E$	$\to_{d\lambda}$	$\lambda\alpha.t^E$	
$\mathsf{F}^n(t_1, \ldots, t_n)^E$	$\to_{dF}$	$\mathsf{F}^n(t_1^E, \ldots, t_n^E)$	if $n \geq 1$
$(ts)^E$	$\to_{d@}$	$t^E s^E$	
$\langle \alpha \mid F\rangle^E$	$\to_{d[]}$	$\langle \alpha \mid F^E\rangle$	if α is bound by F
$\lambda\phi_1$			
Box elimination rules:			
$\lambda\alpha.t^{E_1 \cdot E}$	$\to_{[]\lambda}$	$(\lambda\alpha.t^{E_1})^E$	if $E = \varepsilon$ and $E_1, \alpha > E$
$\mathsf{F}(t_1, \ldots, t_i^E, \ldots, t_n)$	$\to_{[]F}$	$\mathsf{F}(t_1, \ldots, t_i, \ldots, t_n)^E$	
$t^E s$	$\to_{[]@_1}$	$(ts)^E$	
ts^E	$\to_{[]@_r}$	$(ts)^E$	
$(t^E)^F$	$\to_{[]m}$	$t^{E,F}$	
$\langle t \mid \alpha = s^{E_1 \cdot E}, F\rangle$	$\to_{[]_{[]}}$	$\langle t \mid \alpha = s^{E_1}, E, F\rangle$	if $E \neq \varepsilon$ and $E_1, \alpha > E$
Copying:			
t	$\to_{c}$	s	if $\exists$ a variable mapping σ, $s^\sigma \equiv t$
$\lambda\phi_2$			

PROPOSITION 9.22. *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 9.23. *$\lambda\phi_2$ 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:

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

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

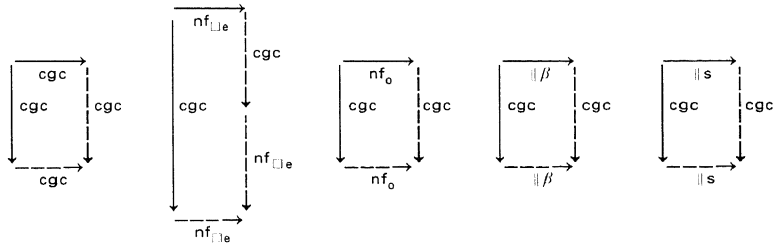
$t \to_{\mathsf{cgc}} s$: if s is obtained from t by performing a copy step followed by the reduction to normal form with respect to garbage collection;

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

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

We next prove the weak confluence diagrams.

$\to_{\mathsf{cgc}}$ and the other rules.



The confluence of copying is shown in [AK96]. Copy does not commute with $\to_{\mathsf{nf}_{\Box\mathsf{e}}}$ because a copy step can turn some cyclic boxes into acyclic boxes, as shown next:

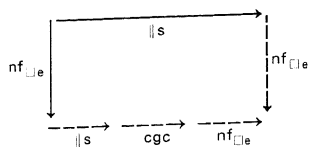
$$t \equiv \langle \alpha \mid \alpha = \underline{\langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha \rangle} \rangle \to_{\mathsf{c}} \langle \alpha \mid \alpha = \underline{\langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha' \rangle}_1,$$
$$\alpha' = \underline{\langle \mathsf{F}\delta \mid \delta = \mathsf{G}\alpha' \rangle}_2 \rangle \equiv s.$$

The underlined cyclic box in t has two descendants in s, of which the one subscripted with one is acyclic.

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

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

The same happens in case of acyclic substitution. In other words, $\rightarrow_{\square_{[\,]}}$ interferes with substitution if $s^{E,F}$ is involved in the substitution. Following a similar argument as in the study of the interaction between $\rightarrow_{d\square}$ and the substitution rules (see the proof of Theorem 9.23), and from the interaction between cgc and $\mathsf{nf}_{\square_e}$ we have the following commuting diagram:



$\rightarrow_{\mathsf{nf}_{\square_e}}$ and $\rightarrow_{\mathsf{nf}_0}$. Let us first analyze each distribution rule.

$\rightarrow_{d\lambda}$.

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

$\rightarrow_{d\omega}$.

$$\begin{array}{ccc} (ts)^E & \xrightarrow[d\omega]{} & t^E s^E \\ \downarrow c & & \downarrow \square_{\omega l} \\ & & (t's^E)^{E'} \\ & & \downarrow \square_{\omega r} \\ & & ((t's)^E)^{E'} \\ & & \downarrow \square m \\ (t's)^{E,E'} & \equiv & (t's)^{E,E'} \end{array}$$

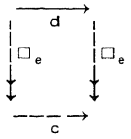
$\rightarrow_{dF}$. Same as the case above.

$\rightarrow_{d\square}$. Let F contain n-equations. Then

$$\begin{array}{ccc} \langle\langle\alpha \mid E\rangle \mid F\rangle & \xrightarrow[d\square]{} & \langle\alpha \mid E^F\rangle \\ \downarrow \square m & & \downarrow \square_e \\ \langle\alpha \mid E, F\rangle & \xrightarrow[c]{} & \langle\alpha \mid E', F_1, \ldots, F_n\rangle \end{array}$$

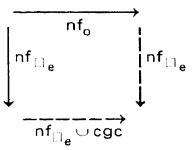
where $\twoheadrightarrow_{\square_e}$ stands for the reduction relation induced by the box elimination rules and garbage collection.

Summarizing, we have:



(9.12)

From (9.12) and the fact that copying commutes with $\twoheadrightarrow_{\square_e}$,

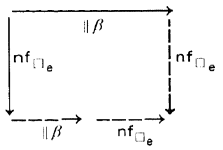


where $\twoheadrightarrow_{\mathrm{nf}_{\square_e} \cup \mathrm{cgc}}$ stands for the reduction relation induced by $\rightarrow_{\mathrm{nf}_{\square_e}}$ and $\rightarrow_{\mathrm{cgc}}$.

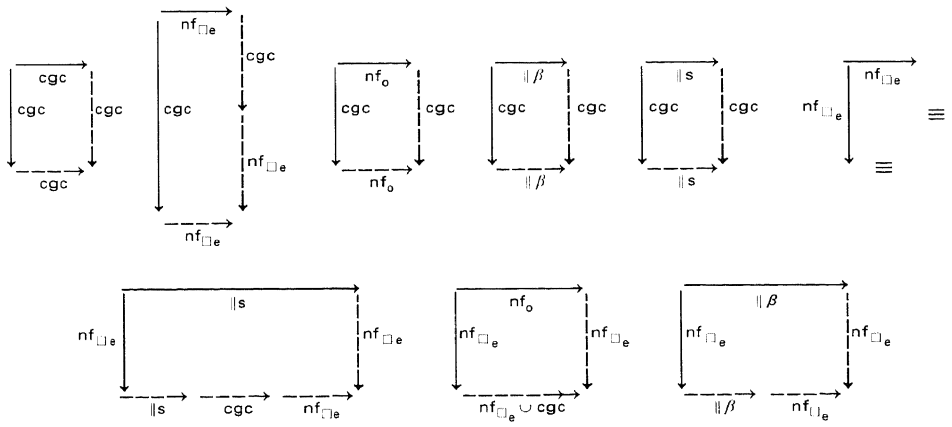
$\rightarrow_{\mathrm{nf}_{\square_e}}$ and $\rightarrow_{\|\beta}$. The only interference is caused by $\rightarrow_{\square\lambda}$:

$$\begin{array}{ccc} (\lambda\alpha.t^{E_1,E})s & \xrightarrow[\beta]{} & \langle t^{E_1,E} \mid \alpha = s\rangle \\ \downarrow \square\lambda & & = \mathrm{nf}\square_e \\ (\lambda\alpha.t^{E_1})^E s & \xrightarrow[\square\omega_1]{} ((\lambda\alpha.t^{E_1})s)^E \xrightarrow[\beta]{} & \langle t^{E_1} \mid \alpha = s\rangle^E. \end{array}$$

Following an argument similar to that in the study of the interaction between $\rightarrow_{\mathrm{d}\omega}$ 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 diagrams:



According to the ordering $\mathsf{nf}_{\Box_e} < \mathsf{cgc} < \mathsf{nf}_o < \|\beta < \|\mathsf{s}$, the above diagrams are decreasing, and thus by Theorem 9.18 $\lambda\phi_2'$ 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 $\lambda\phi_2$. ∎

The presence of both distribution and box elimination rules cause infinite reductions like the following one:

$$(\lambda\alpha.\alpha\delta)\,\gamma \to_\beta \langle \alpha\delta \mid \alpha = \gamma\rangle \to_{\mathrm{d}\omega} \langle \alpha \mid \alpha = \gamma\rangle\langle \delta \mid \alpha = \gamma\rangle \twoheadrightarrow_{\mathrm{gc}}$$
$$\langle \alpha \mid \alpha = \gamma\rangle\, \delta \to_{\Box\omega_1} \langle \alpha\delta \mid \alpha = \gamma\rangle \twoheadrightarrow \cdots.$$

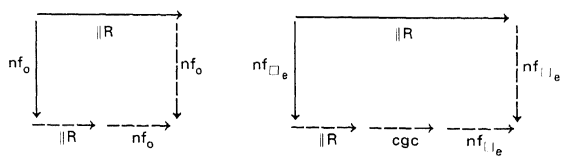
Thus, $\lambda\phi_2$ does not preserve strong normalization on strong normalizing λ-terms. We also point out that the system $\lambda\phi_0 \cup$ *Box elimination* rules $\cup$ *Copying* is confluent.

Remark 9.24. Given a λ-calculus term $M \equiv \lambda x.C[N]$, N is said to be a free expression of M if all free variables of N are free in M. N is said to be a maximal free expression (mfe) of M if M does not contain any other free expression that properly contains N. If we start from a λ-calculus term such that each λ-abstraction does not have trivial mfe's (i.e., different from a variable) then 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 sharing of arguments is enough to capture the amount of sharing offered by Wadsworth's interpreter.

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

THEOREM 9.25. *Let R be an orthogonal term rewriting system. Then $\lambda\phi_2 \cup R$ is confluent.*

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



where $\to_{\|R}$ stands for a complete development of a set of R-redexes. ∎

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

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

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

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

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

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

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

$$l^\sigma \to \langle r \mid x_1 = t_1, \ldots, x_n = t_n \rangle,$$

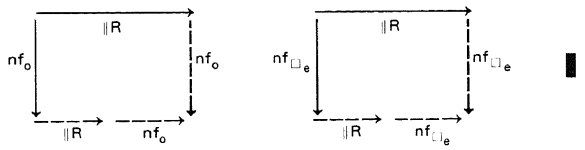
where σ is the mapping $x_1 \mapsto t_1, \ldots, x_n \mapsto t_n$. The alternative we pursue instead is to require the right-hand side of a first-order term rewriting rule to be a $\lambda\phi$-term, which is linear in its free variables. For example, we express the rule $\mathsf{F}(\alpha) \to \mathsf{G}(\alpha,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

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

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

THEOREM 9.26. *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:



10. PREVIOUS WORK

This work follows the tradition of providing calculi that model more closely important practical concerns in a language implementation. In particular, our work has focused on developing a theory able to capture horizontal and vertical sharing in the context of lambda calculus and first-order rewriting. Most of the previous work is concerned with first-order theories [SPvE93]. The operational approach of Barendregt *et al.* [BvEG$^+$87], Smetsers [Sme93], Kennaway *et al.* [KKSdV94], and Farmer *et al.* [FW91, Far90] is based on pointers, redirections, and indirections. The category-oriented approach of Löwe [Löw93], Raoult [Rao84], and Kennaway [Ken87, Ken90] describes graph rewriting in terms of a single or double pushout. The set-theoretic approach of Ariola *et al.* [Ari92, Ari96, AA93, AA95,

AK96] and Raoult *et al.* [RV92] is the approach described in this paper. Typical results are confluence and correctness with respect to either infinitary term rewriting [KKSdV94, Far90] or finite approximations [Ari96].

The issue of lambda calculus and sharing has been addressed by Launchbury [Lau93] and Purushothaman *et al.* [PS92] in an attempt to specify the operational semantics of lazy functional languages such as Haskell [HPJW+92]. Purushothaman *et al.* deal with vertical sharing only. Launchbury's evaluator deals with both kinds of sharing. However, Launchbury does not provide an equational theory, and as such his work is not useful for expressing and reasoning about compiler transformations. Sharing has been studied in the framework of the calculus of explicit substitution by Field [Fie90] and Rose [Ros92b]. Usually this approach to sharing is referred to as the environment model, where an environment is a collection of mappings between variable names and terms. Rose's system allows cyclic structures and will be discussed below.

The issue of sharing has also been studied in the context of optimal (according to Lévy's theory [Lév78]) implementations of λ-calculus. For example, see Mackie [Mac94], Asperti and Laneve [AL94, Lan93], Lamping [Lam90], Kathail [Kat90], and Gonthier *et al.* [GAL92]. In this approach sharing is made explicit by the use of fan-in nodes. Both kinds of sharing are covered and surprisingly the proposed calculi still enjoy confluence. The explanation for this fact is that the mechanism of copying in those calculi is more refined than ours, namely node-by-node. We will discuss the relation with this work in more depth in Section 10.2. We remark that our approach is not optimal.

10.1. Rose's System.

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:

$$S ::= M^{\mu}$$

$$M ::= \alpha \mid (\lambda\alpha.S) \mid (ST)$$

$$\mu ::= \alpha_1 = S_1, \ldots, \alpha_k = S_k.$$

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 4. $\beta\mu$, μ_2, μ_3, and μ_4 can be simulated in $\lambda\phi_2$ as follows:

$\beta\mu$:

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

TABLE IV

Rose's λμ-Calculus

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

μ_2:

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

μ_3:

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

μ_4:

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

Thus, the main difference between $\lambda\phi_2$ and Rose's calculus concerns μ_1, which is absent in $\lambda\phi_2$. 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:

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

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

10.2. Interaction Nets

Differently from the λ-graphs drawn in this paper, a net is an undirected graph, in which the sharing is not represented by multiple pointers to the same node, but by a specific node, called fan-in following Lamping [Lam90]. The fan-in node is drawn as in Fig. 28a; we will often omit the $\circ$ and $\star$ symbols. We will come back later to the explanation of these symbols. In the fan-in node the two nets connected to the higher links share the net connected to the lower link. When the lower link is connected to a lambda-node, the fan-in node is in charge of duplication or copying. A fan-in drawn upside-down is called fan-out, see Fig. 28b. While the fan-in is responsible for sharing, the fan-out is responsible for unsharing. More precisely, the fan-out node allows partial sharing; the net connected to the higher link is shared and is connected to different nets depending on which side ($\circ$ or $\star$) we exit the fan-out node. This partial sharing was first introduced by Lamping [Lam90] and Kathail [Kat90] to provide an optimal (according to Lévy's theory [Lév78]) implementation of λ-calculus, and provides the essential ingredient to solve our counterexamples to confluence. Lastly, following an idea used by Bourbaki in "*Elements de Théorie des Ensembles*" to deal with quantifiers, a variable is represented by a link to the corresponding binding node.

Summarizing, a net for λ-calculus contains the kind of nodes drawn in Fig. 29. Each node has a fixed number of ports. For example, the lambda-node has three ports, connecting the lambda-node to the context, to the bound occurrences, and to the function body. One particular port is called the principal port (indicated with an heavy line). The principal port allows an interaction between the nodes to occur. The last node is the erasure node (see [Mac94]) that is used to represents terms of the form $\lambda x.M$, where the bound variable x does not occur free in M. The terms $\lambda x.x$, $\lambda x.xx$, and $\lambda x.y$ are represented by the nets of Fig. 30. In the following, in drawing nets we may take the liberty of using variables names. Thus we represent the system

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

as in Fig. 31a. Note that we have included a fan-in node between the application and the λy-node even though the λy-node is not shared. This is to capture the fact

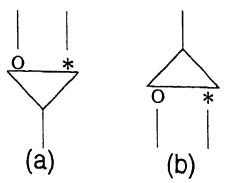


FIG. 28. Fan-in and fan-out nodes.

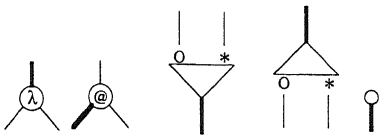


FIG. 29. Nodes of an interaction net for λ-calculus.

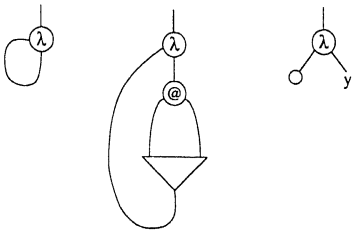


FIG. 30. Interaction nets for $\lambda x.x$, $\lambda x.xx$, and $\lambda x.y$.

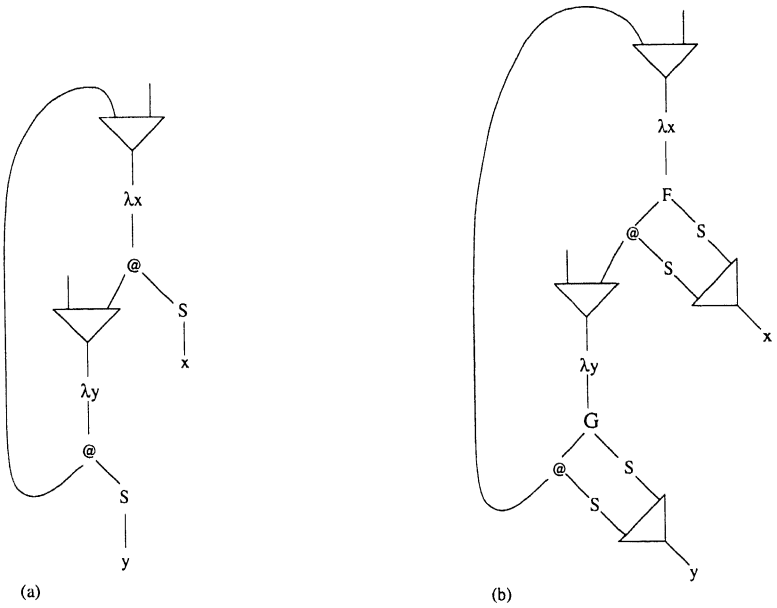


FIG. 31. Cyclic interaction nets.

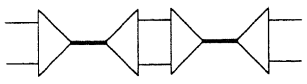


FIG. 32. Interaction net with two interactions.

that $\delta(\mathsf{S}x)$ is an implicit β-redex. We do the same in the representation of the system

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

which is drawn in Figure 31b.

As was said earlier, these nodes (also called agents by Lafont [Laf90]) interact in a very controlled way, namely through the principal port. It is possible to specify an action when an interaction occurs by using rewrite rules, which are restricted to binary interactions. For example, the net of Fig. 32 cannot be the left-hand side of a rule, since it specifies two interactions. Moreover, for each interaction we can specify at most one rewrite rule. These conditions guarantee that interaction nets satisfy the diamond property, as stated in [Laf90].

We can now specify the reduction rules for λ-calculus. First, the β-rule expresses an interaction between the λ and the application node and is drawn as in Fig. 33. The connection of link α to link δ expresses the fact that the root of the redex is over-written by the body of the function. The connection of link x to link γ expresses the fact that the bound variables are replaced by a reference to the argument. For example, the reduction of $(\lambda x.xx)(\lambda x'.x'x')$ is given in Fig. 34. We can formulate this rule in our equational framework by relaxing our scope rules, namely by allowing the body of a lambda abstraction to be spread out through the set of equations. The β-rule then becomes

$$\alpha = (\lambda x.\delta)\gamma \rightarrow \alpha = \delta, x = \gamma,$$

where δ and γ are recursion variables. We can then mimic the reduction of Fig. 34 in our modified equational framework with the following reduction:

$$\begin{array}{lll} \alpha = (\lambda x.\delta)\gamma, & \rightarrow & \alpha = \delta, \\ \delta = xx, & & \delta = xx, \\ \gamma = \lambda x'.\delta', & & x = \gamma, \\ \delta' = x'x' & & \gamma = \lambda x'.\delta', \\ & & \delta' = x'x'. \end{array}$$

Let us now assume there exists an obstacle to the λ-@ interaction, namely, there is a fan-in node between the application and the lambda-node. This corresponds to the situation in our equational framework of having a name associated to the λ-node. Consider the system

$$\langle \alpha \mid \alpha = \lambda x.\delta, \delta = \alpha\gamma, \gamma = x\rangle,$$

in which we assume the variable x is bound by the lambda-node. To make $\alpha\gamma$ into an explicit β-redex we need to apply the substitution operation. Thus,

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

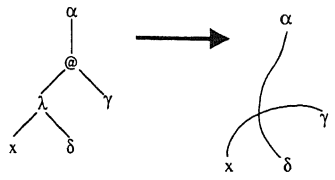


FIG. 33. β-rule.

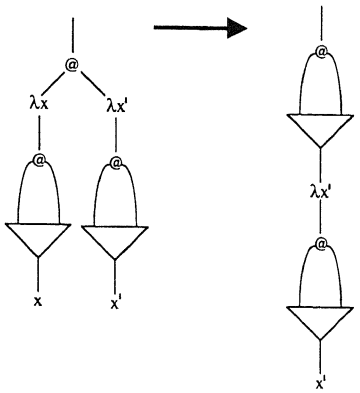


FIG. 34. β-reduction of $(\lambda x.xx)(\lambda x'.x'x')$.

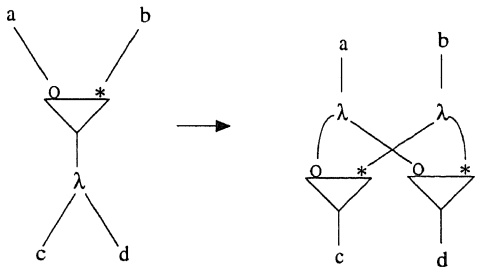


FIG. 35. Fan-in and λ-interaction.

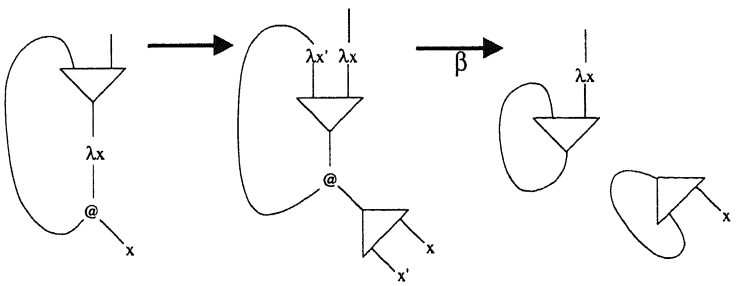


FIG. 36. Fan-in and λ-reduction.

Note that only the lambda-node has been duplicated. However, we now have that variable x is in the scope of two lambdas. This requires the introduction of a mechanism for unsharing x. This is indeed the job of the fan-out node. The right way of performing the above substitution should be

$$\langle \alpha \mid \alpha = \lambda x.\delta,\ \delta = \alpha\gamma,\ \gamma = x \rangle \to_s \langle \alpha \mid \alpha = \lambda x.\delta,\ \delta = (\lambda x'.\delta)\gamma,\ \gamma = \mathsf{Fan\text{-}out}(x', x) \rangle. \tag{10.13}$$

The substitution operation is captured in interaction nets by the rules expressing the interaction between a fan-in and a λ (see Fig. 35). By crossing the lambda-node the fan-in node is duplicated. One copy is in charge of duplicating the lambda-body and the other one is responsible for creating two copies of the bound variable. The substitution given in 10.13 is displayed in Fig. 36. (Note that the net on the right consists of two connected nets, since x is a link to the λ-node.) This rule outlines a very important difference between interaction nets and our equational framework. The copying necessary to implement the β-rule is done lazily in the interaction nets approach, namely, it is done node-by-node. Instead, in our framework it is done at once. In fact, corresponding to the reduction in question we would have

$$\langle \alpha \mid \alpha = \lambda x.\alpha x \rangle \to_s \langle \alpha \mid \alpha = \lambda x.(\lambda x.\alpha x)x \rangle \to_\beta \langle \alpha \mid \alpha = \lambda x.\alpha x \rangle \to_s \cdots.$$

We finally have the rules that deal with fan-in and fan-out nodes. If the fan-in and fan-out nodes match, that is the fan-out node is the one introduced by the corresponding fan-in node, then they cancel each other out (see the rule on the left of Fig. 37). Otherwise, both fan-in and fan-out nodes are duplicated (see the rule on the right of Fig. 37). In order to keep track of the matching between fan-in and fan-out nodes, fan nodes need to be labelled with an index which varies during reduction under the control of some additional nodes called brakets. We do not present this additional mechanism here but refer to [GAL92, Mac94, AL94, Lam90].

In [Mac94] it is mentioned that our counterexamples disappear in the framework of interaction nets. We are now ready to show in detail how this happens. In Fig. 38 we show the reduction corresponding to the counterexample of Section 4.

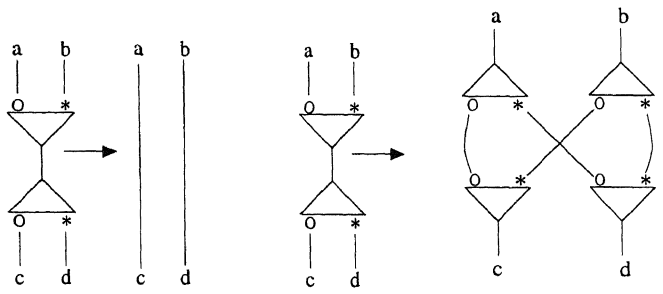


FIG. 37. Fan-in and fan-out rules.

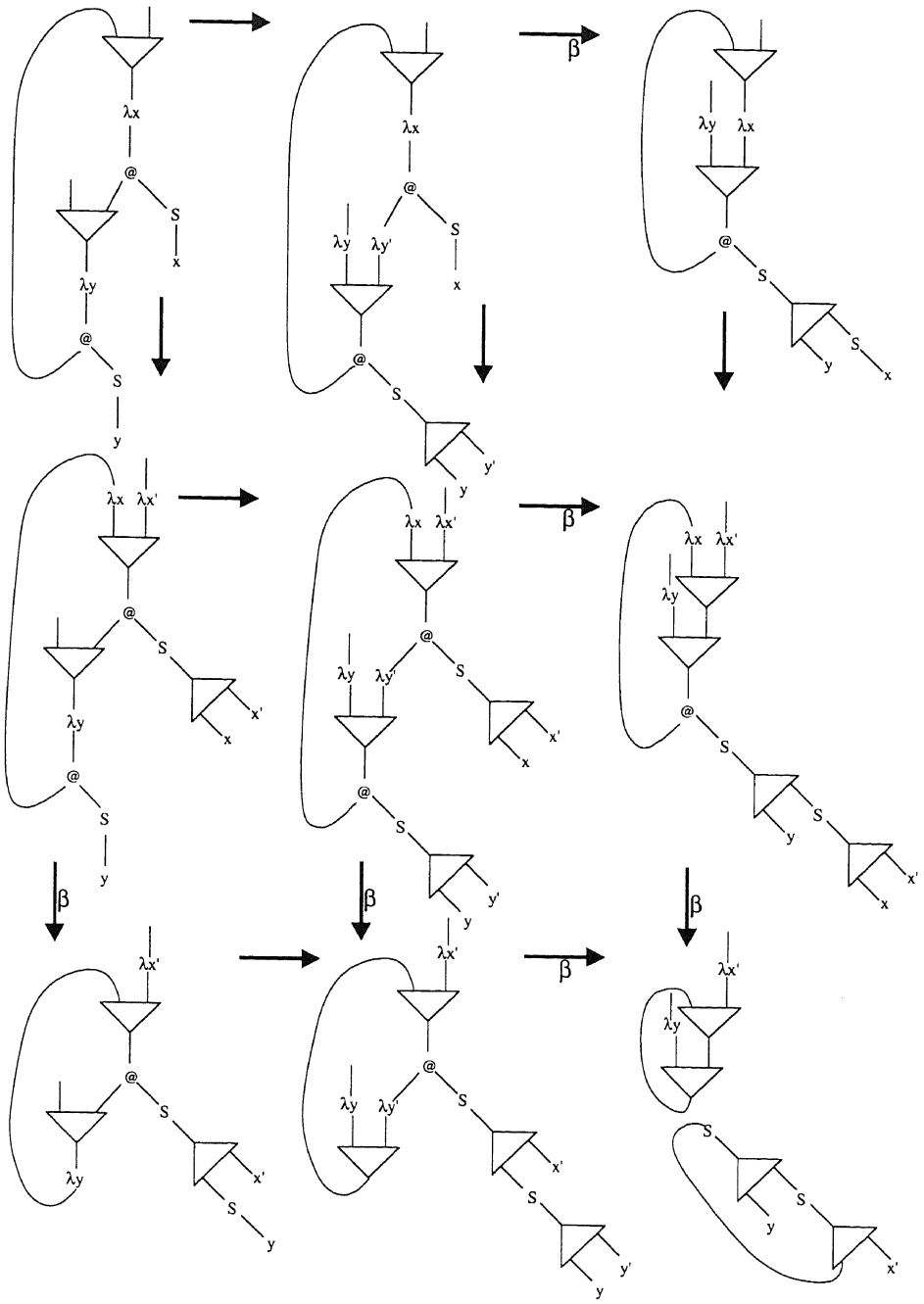


FIG. 38. First counterexample in interaction nets.

With the introduction of the fan-out nodes we solve the out of sync phenomenon since we are no longer required to copy an even number of S's.

Let us now turn to the third counterexample described in Section 6, which is given in Fig. 39. The common net should thus correspond to an irregular tree. This translation, also called read back semantics, is explained next. Let us call the

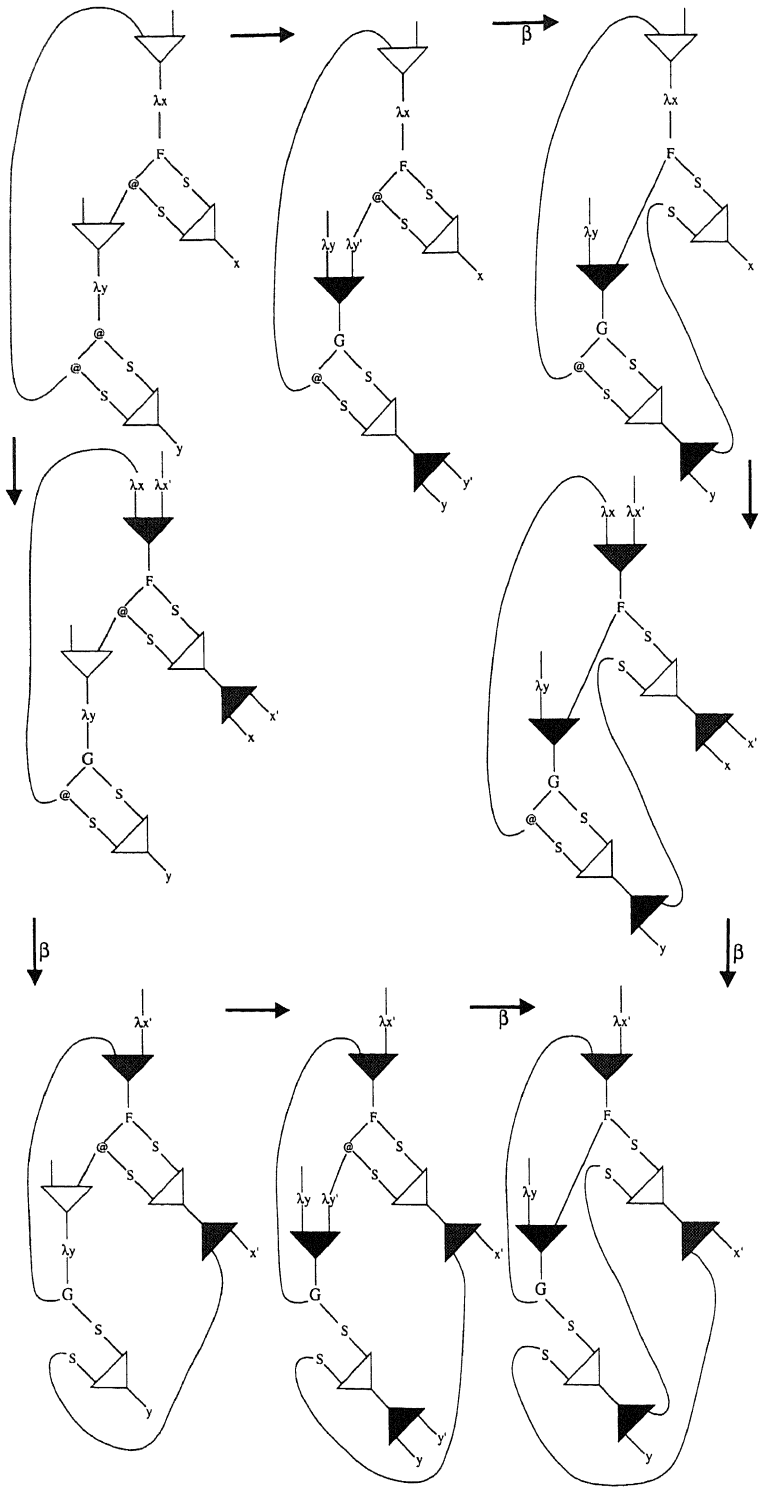


FIG. 39. Third counterexample in interaction nets.

top-most fan-in node the red fan-in and the lower one the blue fan-in. In the figure we have indicated the corresponding fan-outs. Note that at this point the symbols ◦ and ★ are significant. For simplicity we refer to ◦ as L (left) and to ★ as R (right). In the read back procedure (called unwinding in this paper) we make use of a stack to remember which port of the fan-in we enter from. We start by generating a $\lambda x'$-node. When we enter the fan-in node we push on the red-stack the symbol R. We connect the $\lambda x'$-node to the node labelled F. Since F has two arguments we duplicate the red stack; one is used to generate the first argument and the other one is used for generating the second argument. Let us continue with the second argument. We generate an S and we go through a fan-in node, but since there are no associated fan-out nodes we do not save anything on the stack. We then reach the red fan-out, from which we must exit from the port the associated fan-in was entered from. This information is saved on the red stack of the second argument. Since on the top of the stack we read R we exit the fan-out from the right port and we thus generate x' and pop R from the red stack. Since we have reached a bound variable it means that we have finished generating the second argument of F. We now go back to the generation of the first argument of *F*. We go through the blue fan-in. We thus push R on the blue stack. We then connect the first argument to a G. As before since G has two arguments we duplicate both the red and blue stacks. We note that both stacks now contain R. We continue with the second argument of G. We generate an S, we then exit the blue fan-out with an S and the red fan-out with an x', indicating that we have finished with the second argument of G. With respect to the first argument we first push L on the red stack. We connect the G to an F and duplicate the stacks. On the second argument of F we connect the F to an S and then exit the red fan-out from the L port (and pop the red stack). This means that we generate one S. We then exit the blue fan-out from the R port (the blue stack is now empty), thus generating one more S. We finally exit the red fan-out from the R port. This completes the generation of this argument. At this point we have the tree drawn in Fig. 40. The rest is generated in a similar way.

In conclusion, our counterexamples to confluence disappear in this framework, however, at the expense of greater complexity. Moreover, the correctness of this approach has only been shown with respect to ordinary λ-calculus; thus it would be interesting to prove that correctness also holds for cyclic graphs.

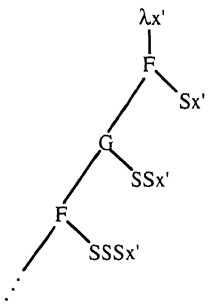


FIG. 40. Partial tree.

CONCLUSIONS AND FUTURE DIRECTIONS

We have defined a series of calculi as extensions of the λ-calculus with the aim of providing systems where it is possible to model sharing and cyclic structures. The motivation for this work came from the desire to provide a unifying framework for reasoning about execution, compilation, and optimization of programs. In these three areas sharing and cycles are ubiquitous; they occur after parsing, in the intermediate program representation (language), and during program execution.

The focus of this paper has been on developing calculi that enjoy the confluence property. As such, the resulting calculi fail to capture program transformations that deal with mutually recursive functions. Our next step is to study calculi that have a more liberal view of rewriting, i.e., substitutions can occur on a cycle. This involves the introduction of a more abstract notion of confluence. Whilst confluence guarantees unicity of normal forms, the new notion of confluence should guarantee unicity of infinite normal forms. These calculi should correspond to the intermediate languages used in the compilation of the functional core of both strict and non-strict languages. We intend to make use of these calculi in studying the effects of different strategies on both the time and space behavior of programs and relating them to current optimizations, including loop transformations.

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 with 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. We will also consider the rewriting of disconnected graphs, which, as shown by Pinter *et al.* [PP94], is useful for detecting parallelizable program structures in sequential programs.

ACKNOWLEDGMENTS

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 Vrije Universiteit Amsterdam. Zena Ariola thanks both CWI and the Vrije Universiteit for making 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 Femke van Raamsdonk and Vincent van Oostrom for introducing us to interaction nets, Stefan Blom for scrutinizing several proofs, and Amr Sabry for stimulating discussions about a draft of this paper. We also thank the anonymous referees for their useful comments.

Received April 3, 1996; final manuscript received May 13, 1997

REFERENCES

[AA93] Ariola, Z. M., and Arvind (1993), Graph rewriting systems for efficient compilation, *in* "Term Graph Rewriting: Theory and Practice" (M. R. Sleep, M. J. Plasmeijer, and M. C. D. J. van Eekelen, Eds.), pp. 77–90, Wiley, New York.

[AA95] Ariola, Z. M., and Arvind (1995), Properties of a first-order functional language with sharing, *Theoret. Comput. Sci.* **146**, 69–108.

[ACCL91] Abadi, M., Cardelli, L., Curien, P.-L., and Lévy, J.-J. (1991), Explicit substitutions, *J. Funct. Programming* **4**, No. 1, 375–416.

[AF] Ariola, Z. M., and Felleisen, M. (1997), The call-by-need lambda calculus, *J. Funct. Programming*, **7**, No. 3. [Also Technical Report CIS-TR-96-97]

[AFM+95] Ariola, Z. M., Felleisen, M., Maraist, J., Odersky, M., and Wadler, P. (1995), The call-by-need lambda calculus, *in* "Proceedings, ACM Conference on Principles of Programming Languages," pp. 233–246.

[AK94] Ariola, Z. M., and Klop, J. W. (1994), Cyclic lambda graph rewriting, *in* "Proceedings, Ninth Symposium on Logic in Computer Science (LICS'94), Paris, France," pp. 416–425.

[AK96] Ariola, Z. M., and Klop, J. W. (1996), Equational term graph rewriting, *Fund. Inform.* **26**, No. 3, 4, 207–240. [Extended version: CWI Report CS-R9552]

[AKK+94] Ariola, Z. M., Klop, J. W., Kennaway, J. R., de Vries, F. J., and Sleep, M. R. (1994), Syntactic definitions of undefined: On defining the undefined, *in* "Proceedings, TACS 94, Sendai, Japan," pp. 543–554.

[AKP84] Arvind, Kathail, V., and Pingali, K. (1984), Sharing of computation in functional language implementations, *in* "Proceedings, International Workshop on High-Level Computer Architecture."

[AL94] Asperti, A., and Laneve, C. (1994), Interaction systems. I. The theory of optimal reductions, *Math. Structures Comp. Sci.* **4**, 457–504.

[App92] Appel, A. (1992), "Compiling with Continuations," Cambridge Univ. Press.

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

[Ari96] Ariola, Z. M. (1996), Relating graph and term rewriting via Böhm models, *Appl. Algebra Engrg. Comm. Comput.* **7**, No. 5, 401–426.

[Bar84] Barendreght, H. P. (1984), "The Lambda Calculus: Its Syntax and Semantics," North-Holland, Amsterdam.

[BD77] Burstall, R., and Darlington, J. (1977), A transformation system for developing recursive programs, *J. Assoc. Comput. Mach.* **24**, 44–67.

[BvEG+87] Barendregt, H. P., van Eekelen, M. C. J. D., Glauert, J. R. W., Kennaway, J. R., Plasmeijer, M. J., and Sleep, M. R. (1987), Term graph rewriting, *in* "Proceedings of the Conference on Parallel Architecture and Languages Europe (PARLE '87), Eindhoven, The Netherlands" (J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, Eds.), Lecture Notes in Computer Science, Vol. 259, pp. 141–158, Springer-Verlag, Berlin/New York.

[CR90] Clinger, W., and Rees, J. (1990), "Revised Report on the Algorithmic Language Scheme," Technical Report CIS-TR-90-02, University of Oregon.

[Cur93] Curien, P.-L. (1993), "Categorical Combinators, Sequential Algorithms, and Functional Programming," 2nd ed., Birkhäuser, Basel.

[DJ90] Dershowitz, N., and Jouannaud, J. P. (1990), Rewrite systems, *in* "Handbook of Theoretical Computer Science" (J. van Leeuwen, Ed.), Vol. B, pp. 243–320, Elsevier/MIT Press, Amsterdam, New York/Cambridge, MA.

[Far90] Farmer, W. M. (1990), A correctness proof for combinator reduction with cycles, *ACM Trans. Program. Lang. and Systems* **12**, No. 1, 123–134.

[Fie90] Field, J. (1990), On laziness and optimality in lambda interpreters: Tools for specification and analysis, *in* "Proceedings, Conference on Principles of Programming Languages, San Francisco."

[FW91] Farmer, W. M., and Watro, R. J. (1991), Redex capturing in term graph rewriting, *in* "Proc. 4th International Conference on Rewriting Techniques and Applications (RTA-91), Como, Italy" (R. V. Book, Ed.), Lecture Notes in Computer Science, Vol. 488, pp. 13–24, Springer-Verlag, Berlin/New York.

[GAL92] Gonthier, G., Abadi, M., and Lévy, J.-J. (1992), The geometry of optimal lambda reductions, *in* "Proceedings, ACM Conference on Principles of Programming Languages."

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

[HM76] Henderson, P., and Morris, J. H. (1976), A lazy evaluator, *in* "Proceedings, ACM Conference on Principles of Programming Languages."

[HPJW+92] Hudak, P., Peyton Jones, S., Wadler, P., Boutel, B., Fairbairn, J., Fasel, J., Hammond, K., Hughes, J., Johnsson, T., Kieburtz, D., Nikhil, R., Partain, W., and Peterson, J. (1992), Report on the programming language Haskell, *ACM SIGPLAN Notices* **27**, No. 5, 1–64.

[Hug82] Hughes, R. J. M. (1982), Super-combinators, *in* "Proceedings of Lisp and Functional Programming."

[JGS93] Jones, N. D., Gomard, C., and Sestoft, P. (1993), "Partial Evaluation and Automatic Program Generation," Prentice-Hall, New York.

[Joh85] Johnsson, T. (1985), Lambda lifting: Transforming programs to recursive equations, *in* "Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture, Nancy, France," Lecture Notes in Computer Sciences, Vol. 201, Springer-Verlag, Berlin/New York.

[Kat90] Kathail, V. K. (1990), "Optimal Interpreters for Lambda-Calculus Based Functional Languages," Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, MIT.

[Ken87] Kennaway, J. R. (1987), On graph rewriting, *Theoret. Comput. Sci.* **52**, 37–58.

[Ken90] Kennaway, J. R. (1990), Graph rewriting on some categories of partial morphisms, *in* "Proceedings of the 4th International Workshop on Graph Grammars and their Application to Computer Science, Bremen, Germany," Lecture Notes in Computer Sciences, Vol. 532, pp. 490–504, Springer-Verlag, Berlin/New York.

[KKSdV94] Kennaway, J. R., Klop, J. W., Sleep, M. R., and de Vries, F. J. (1994), On the adequacy of graph rewriting for simulating term rewriting, *Trans. Program. Lang. and Systems* **16**, No. 3, 493–523.

[KKSdV95a] Kennaway, J. R., Klop, J. W., Sleep, M. R., and de Vries, F. J. (1995), Infinitary lambda calculus, *in* "Proceedings, Rewriting Techniques and Applications, Kaiserslautern."

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

[Klo] Klop, J. W. (1980), "Combinatory Reduction Systems," Ph.D. thesis, Mathematical Centre Tracts **127**, CWI, Amsterdam.

[Klo92] Klop, J. W. (1992), Term rewriting systems, *in* "Handbook of Logic in Computer Science" (S. Abramsky, D. Gabbay, and T. Maibaum, Eds.), Vol. II, pp. 1–116, Oxford Univ. Press, London.

[KvOvR93] Klop, J. W., van Oostrom, V., and van Raamsdonk, F. (1993), Combinatory reduction systems: Introduction and survey, *Theoret. Comput. Sci.* **121**, No. 1, 2, 279–308. [A collection of contributions in honour of Corrado Böhm on the occasion of his 70th birthday, (M. Dezani-Ciancaglini, S. Ronchi Della Rocca, and M. Venturini-Zilli, Guest Eds.)]

[Laf90] Lafont, Y. (1990), Interaction nets, *in* "Proceedings, ACM Conference on Principles of Programming Languages, San Francisco."

[Lam90] Lamping, J. (1990), An algorithm for optimal lambda calculus reduction, *in* "Proceedings, ACM Conference on Principles of Programming Languages, San Francisco."

[Lan93] Laneve, C. (1993), "Optimality and Concurrency in Iteration Systems," Ph.D. thesis, University of Pisa.

[Lau93] Launchbury, J. (1993), A natural semantics for lazy evaluations, *in* "Proceedings, ACM Conference on Principles of Programming Languages," pp. 144–154.

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

[Lév78] Lévy, J.-J. (1978), "Réductions correctes et optimales dans le lambda-calcul," Ph.D. thesis, Université Paris VII.

[Löw93] Löwe, M. (1993), Algebraic approach to single pushout graph transformation, *Theoret. Comput. Sci.* **109**, 181–224.

[Mac94] Mackie, Ian Craig (1994), "The Geometry of Implementation," Ph.D. thesis, University of London.

[Nik91] Nikhil, R. S. (1991), "Id (Version 90.1) Reference Manual," Technical Report 284-2, MIT Laboratory for Computer Science, Cambridge, MA.

[PJ87] Peyton Jones, S. L. (1987), "The implementation of Functional Programming Languages," Prentice Hall International, Englewood Cliffs, NJ.

[PP94] Pinter, S. S., and Pinter, R. Y. (1994), Program Optimization and parallelization, *ACM Trans. Program. Lang. Systems* **16**, No. 3, 305–327.

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

[Rao84] Raoult, J. C. (1984), On graph rewritings, *Theoret. Comput. Sci.* **32**, 1–24.

[Ros92a] Rosaz, J. G. (1992), Taming the Y operator, *in* "Proceedings of Lisp and Functional Programming," pp. 226–234.

[Ros92b] Rose, K. H. (1992), Explicit cyclic substitutions, *in* "Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems (CTRS-92), Pont-à-Mousson, France" (M. Rusinowitch and J. L. Rémy, Eds.), Lecture Notes in Computer Science, Vol. 656, pp. 36–50, Springer-Verlag, Berlin/New York.

[RV92] Raoult, J.-C., and Voisin, F. (1992), "Set-Theoretic Graph Rewriting," Technical Report 1665, INRIA Rapport de Recherche.

[Sme93] Smetsers, J. E. W. (1993), "Graph Rewriting and Functional Languages," Ph.D. thesis, University of Nijmegen.

[SPvE93] Sleep, M. R., Plasmeijer, M. J., and van Eekelen, M. C. D. J., Eds. (1993), "Term Graph Rewriting: Theory and Practice," Wiley, New York.

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

[vO94] van Oostrom, V. (1994), "Confluence for Abstract and Higher-Order Rewriting," Ph.D. thesis, Vrije Universiteit.

[Wad71] Wadsworth, C. (1971), "Semantics and Pragmatics of the Lambda-Calculus," Ph.D. thesis, University of Oxford.

[Wad90] Wadler, P. (1990), Deforestation: Transforming programs to eliminate trees, *Theoret. Comput. Sci.* **73**, 231–248.