Explicit Cyclic Substitution

Kristoffer Høgsbro Rose

DIKU, University of Copenhagen, Universitetsparken 1,
DK–2100 København Ø, Denmark†

E-mail: kris@diku.dk

March, 1993

Abstract. The untyped λ-calculus enriched with local possibly recursive definitions, *e.g.*, **let** and **letrec** of functional programming languages, is discussed. It is shown how it may be modeled using conditional term rewriting systems defining "explicit acyclic and cyclic substitution" calculi that generalise the explicit substitution calculus of Abadi, Cardelli, Curien, and Lévy [1] in two directions: the systems model traditional λ-calculus with variables (thus not restricted to closed terms), and the cyclic calculus may model recursion directly.

1 Introduction

In this section we motivate why we study substitution encoded explicitly in the form of conditional term rewrite system (*CTRS*) and then relate the results in the paper to current work in rewriting systems and semantics of programming languages. Last we give an overview of the remainder of the paper.

Motivation

Most programming languages include some form of 'local definition' that makes sharing explicit. For example, $(2^3 + 3^2) \times (2^3 - 3^2)$ can be computed by the

Invited for submission to special issue of Journal of Symbolic Computation on Conditional Term Rewriting Systems; also available as DIKU semantics note D–166. Presentation [18] supported by the Danish Research Council (SNF) project DART.
Key words and phrases. Substitution, recursion, λ-calculus.
†*Current address*: Dept. of Computer Science, Chalmers University of Technology, S–412 96 Gothenburg, Sweden.

following expression in a generic functional language with explicit sharing of the subexpressions 2^3 and 3^2:

$$\textbf{let } a = 2^3 \textbf{ and } b = 3^2 \textbf{ in } (a+b) \times (a-b)$$

It is clear that the two expressions should compute the same value in any reasonable programming language (where 'reasonable' is usually taken to mean that the language should obey the "referential transparency" principle in one form or the other [20]).

But if we wish to give the two expressions a *rewrite semantics* in the form of a CTRS describing how to obtain this value *operationally*, then we can observe the rewrite sequences, hence the sharing matters because the two expressions rewrite in a different number of steps. For example the unshared version of the term above may rewrite in the following seven steps (where we underline the next subexpression to be reduced):

$$\begin{aligned}
(\underline{2^3} + 3^2) \times (2^3 - 3^2) &\Rightarrow (8 + \underline{3^2}) \times (2^3 - 3^2) \\
&\Rightarrow \underline{(8+9)} \times (2^3 - 3^2) \\
&\Rightarrow 17 \times (\underline{2^3} - 3^2) \\
&\Rightarrow 17 \times (8 - \underline{3^2}) \\
&\Rightarrow 17 \times \underline{(8-9)} \\
&\Rightarrow \underline{17 \times (-1)} \\
&\Rightarrow -17
\end{aligned}$$

In contrast the shared version may rewrite in five steps:

$$\begin{aligned}
&\textbf{let } a = \underline{2^3} \textbf{ and } b = 3^2 \textbf{ in } (a+b) \times (a-b) \\
&\quad \Rightarrow \textbf{let } a = 8 \textbf{ and } b = \underline{3^2} \textbf{ in } (a+b) \times (a-b) \\
&\quad \Rightarrow \textbf{let } a = 8 \textbf{ and } b = 9 \textbf{ in } \underline{(a+b)} \times (a-b) \\
&\quad \Rightarrow \textbf{let } a = 8 \textbf{ and } b = 9 \textbf{ in } 17 \times \underline{(a-b)} \\
&\quad \Rightarrow \textbf{let } a = 8 \textbf{ and } b = 9 \textbf{ in } \underline{17 \times (-1)} \\
&\quad \Rightarrow -17
\end{aligned}$$

Quite similarly we may consider languages with local *recursive* definitions although here we face the difficulty that the unshared version of an expression with a cyclic definition may be infinite. For example,

$$\begin{aligned}
\mathit{fib}(n) = \ &\textbf{letrec } f(n_0, n_1, n') = \textbf{if } n' = 0 \textbf{ then } n_0 \\
&\qquad\qquad\qquad\qquad \textbf{else } f(n_1, n_0 + n_1, n' - 1) \\
&\textbf{in } f(0, 1, n)
\end{aligned}$$

defines $\mathit{fib}(n)$ as the nth fibonacci number for positive n using the local recursive

function f. Here is a way to compute the third number:

$$
\begin{array}{l}
\mathit{fib}(3) \\
\quad \Rightarrow \textbf{letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad \textbf{in } f(0, 1, 3) \\
\quad \Rightarrow \textbf{if } 3 = 0 \textbf{ then } 0 \\
\qquad \textbf{else letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad\qquad \textbf{in } f(1, 1, 2) \\
\quad \Rightarrow \textbf{letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad \textbf{in } f(1, 1, 2) \\
\quad \Rightarrow \textbf{if } 2 = 0 \textbf{ then } 1 \\
\qquad \textbf{else letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad\qquad \textbf{in } f(1, 2, 1) \\
\quad \Rightarrow \textbf{letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad \textbf{in } f(1, 2, 1) \\
\quad \Rightarrow \textbf{if } 1 = 0 \textbf{ then } 1 \\
\qquad \textbf{else letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad\qquad \textbf{in } f(2, 3, 0) \\
\quad \Rightarrow \textbf{letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad \textbf{in } f(2, 3, 0) \\
\quad \Rightarrow \textbf{if } 0 = 0 \textbf{ then } 2 \\
\qquad \textbf{else letrec } f(n_0, n_1, n') \;=\; \textbf{if } n' = 0 \textbf{ then } n_0 \textbf{ else } f(n_1, n_0 + n_1, n' - 1) \\
\qquad\qquad \textbf{in } f(3, 5, -1) \\
\quad \Rightarrow 2
\end{array}
$$

where each step either "unfolds" one application of f or selects the appropriate branch of an **if** choice.

Two things are essential here. First, we "push" the **letrec** definition "inwards" in the term for each unfolding—this is important to keep the recursive definition local to where it is used to make it possibe to dispose of it when unfolding. Second, computation does not proceed with $f(3, 5, -1)$, of course, since that computation would loop indefinitely. In practice this is done by providing a "reduction strategy" showing that one particular kind of reduction sequence will not give problems; in the example we used an 'outermost' strategy.

The purpose of this paper is to treat **let** and **letrec** in the λ-calculus tradition and prove the more general *confluence* property which implies that no matter how silly we reduce it will always be possible to reach the result—if any—by further reduction. Since **let** and **letrec** represent sharing this makes it possible to choose how sharing should be performed.

Contribution & Relation to other work

This paper combines two traditions: conceiving of the λ-calculus as a CTRS and considering functional programming languages as realisations of "enriched" λ-calculi. This gives two theories described by CTRS—$\boldsymbol{\lambda\sigma}$ and $\boldsymbol{\lambda\mu}$—which make

it possible to investigate the sharing properties of different reduction sequences.

The treatment of **let** in $\boldsymbol{\lambda\sigma}$ is a generalisation of the untyped "explicit substitution" calculus of Abadi, Cardelli, Curien, and Lévy [1] to the full λ-calculus with variables and open terms; the treatment of **letrec** by "explicit cyclic substitution" in $\boldsymbol{\lambda\mu}$ is new. The presentation makes the correspondence between the presented CTRS and 'real' functional programming languages clearer, and makes it possible to express and investigate properties of *bindings*, *e.g.*, scoping rules, both in the simple and recursive case.

The method of 'substitution pushing' that we have chosen was inspired by the "Call-by-Mix" strategy of Grue [8]. The technique is closely related to and influenced by *term graph rewriting* [5, 19]. The idea of using term graph rewriting in models of the λ-calculus originates with Wadsworth [22] and has been investigated by several authors, notably Staples [21]. We comment further on this in the conclusion. It may be seen as a formalisation of the idea of *implementing* recursion by cyclic structures that is part of the folklore of computer science [15, 9]. It is interesting that in spite of this the *semantic* properties of the idea have not been investigated in depth explicitly—perhaps this is because the extensional behaviour of cyclic structure and fixed point induction are so similar (as long as only terminating computations are considered interesting). This is why the chosen approach is similar to the "enriched λ-calculi" used to describe and implement functional programming languages [11, 17], and to the "λ_{B}-calculus" graph reduction system of Ariola and Arvind [3] and other graph rewriting systems in the intent to support mutual recursion directly. However, it differs in that there is no implied underlying 'execution model' or 'evaluation strategy'—the descriptions presented here are pure CTRS.

Overview of the paper

In §2 we summarize the λ-calculus using a standard implicit notion of substitution. In §3 we then use this to discuss **let** and the relation between interpretations based on simulation through the λ-calculus and using explicit acyclic substitutions. This is generalised in §4 where we define explicit cyclic substitutions that can model **letrec** in a similar way. Finally we conclude in §5 and give directions for future work.

2 Implicit Substitution and β-reduction

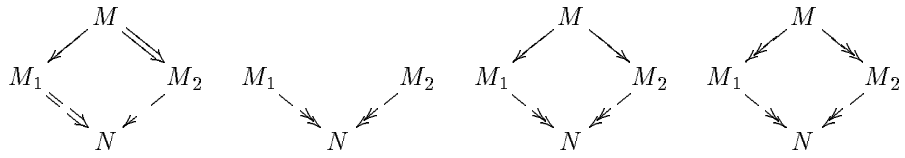
This section summarizes the λ-calculus and traditional λ-reduction based on "implicit" substitutions, *i.e.*, on substitution as metanotation. We define the set Λ of "opaque" λ-terms modulo renaming, and equipped with acyclic simultaneous substitution, β-reduction, and with the usual theorems in the tradition of [4]. Finally we show that 'ordinary' substitution and the one we will use are equivalent in the λ-calculus.

We will make use of fundamental relational concepts as found in [10]:

2.1 Notation. A *reduction step* relation $\rightarrow$ over T is a subset of $T \times T$ that is *stable* and *compatible* in T (this simply means that the reduction step can be applied anywhere in terms).

Given the reduction step relation $\rightarrow$ and let M, N range over terms. *Reduction* $\twoheadrightarrow$ is $\rightarrow$'s transitive, reflexive closure. *Conversion* $\twoheadleftarrow\!\!\twoheadrightarrow$ is $\rightarrow$'s transitive, reflexive, and symmetric closure. M is a *redex* iff $\exists N : M \rightarrow N$, otherwise it is a *normal form*, $\rightarrow$-nf(M). *Convergence* $\twoheadrightarrow\!\!\!\!\mid$ is the restriction to convergent reductions defined by $M \twoheadrightarrow\!\!\!\!\mid N$ iff $M \twoheadrightarrow N$ and $\rightarrow$-nf(N).

Properties will often be expressed by diagram completion, *i.e.*, assertions on the form "assume the solid part of this diagram is satisfied then the entire diagram (including dashed parts) may be satisfied." Here are some such concepts:



Confluent relations are also said to be *Church-Rosser*. Finally, $\rightarrow$ is *strongly normalising* (or *noetherian*) if there are no infinite sequences $M \rightarrow M_1 \rightarrow \ldots$ in it.

Remark. Other notions of reduction are possible, notably by restricting the stability and compatibility requirement, *cf.* Abramsky [2]. A treatment of this is outside the scope of this paper.

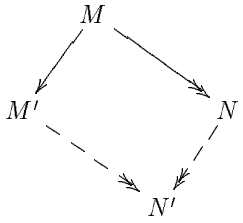
Some general properties apply to reductions and will be used in the following:

2.2 Propositions.

(i) *A strongly normalising relation is confluent iff it is locally confluent* [10, lemma 2.4].

(ii) *Hindley-Rosen lemma: If two reduction relations are confluent and commute, then the transitive reflexive closure of their union is confluent* [4, proposition 3.3.5].

(iii) *Strip lemma induction: Any reduction relation satisfying*



is confluent [10, lemma 2.3].

λ-calculus

The following are similar to the definitions found in [4] except that we define variable renaming separately and allow several variables to be 'simultaneously' substituted.

2.3 Definition. Assume a set $\mathbb{V}$ of variables (ranged over by $x, y, z, \ldots$).

(i) The *λ-terms* (ranged over by $M, N, P, \ldots$) are defined inductively by

$$M \quad ::= \quad x \quad | \quad (\lambda x.M) \quad | \quad (MN)$$

We omit outermost ()s and ()s and λs that may be placed using the following rules: (a) abstraction associates to the right, (b) application associates to the left, and (c) application takes precedence. *E.g.*, $\lambda xy.MNP$ denotes $(\lambda x.(\lambda y.((MN)P)))$.

(ii) A *bound variable* x is x when occuring inside M of $\lambda x.M$.

(iii) The *free variable set* of a term contains those variables that are not bound; inductively:

$$\begin{aligned} \mathrm{FV}(x) &= \{x\} \\ \mathrm{FV}(\lambda x.M) &= \mathrm{FV}(M) \setminus \{x\} \\ \mathrm{FV}(MN) &= \mathrm{FV}(M) \cup \mathrm{FV}(N) \end{aligned}$$

(iv) A term with a *renaming* postfix $[x := y]$ denotes the term obtained by changing 'free occurrences' of x in it to y, inductively specified by:

$$\begin{aligned} x\,[x := y] &\equiv y \\ z\,[x := y] &\equiv z && \text{if } z \neq x \\ (!) \qquad (\lambda z.M)\,[x := y] &\equiv \lambda z'.\, M[z := z'][x := y] && \text{with } z' \notin \mathrm{FV}(xyM) \\ (MN)\,[x := y] &\equiv M[x := y]\,N[x := y] \end{aligned}$$

(v) *Equivalence* of terms means "syntactic equality modulo renaming" (*aka* "α-equivalence"), is denoted ≡, and defined inductively by

$$\begin{aligned} x &\equiv x \\ \lambda x.M &\equiv \lambda y.N && \text{if } M[x := y] \equiv N \\ MN &\equiv PQ && \text{if } M \equiv P \wedge N \equiv Q \end{aligned}$$

(vi) Λ is the set of *opaque* λ-terms, *i.e.*, the λ-terms modulo equivalence: $\Lambda = \lambda\text{-}terms/\equiv$. Any concrete λ-term representing an object in Λ is called a *representative* of that term.

The definition of renaming above avoids 'variable capture,' *i.e.*, changing a free variable into a bound variable, by requiring an extra renaming in (!).

The use of opaque terms means that λ-reduction may be done naïvely provided we adhere to the following.

2.4 Convention. We will always pick a representative of an opaque λ-term where bound variables in distinctly named subterms are distinct from each other and from all free variables. This is the same as Barendregt's *variable convention* [4, convention 2.1.13].

With this convention we are ready for definitions of simultaneous substitution based on the 'variable substitution' (renaming) of 2.3(iv).

2.5 Definitions.

(i) A *binder* is a pair $y := Q$ of a variable and a term. A *binder set* is an unordered collection of the form $y_i := Q_i, y_{i+1} := Q_{i+1}, \ldots, y_k := Q_k$ for integers $0 \leq i \leq k$ where the ys are distinct. We abbreviate binder sets as $y_i := Q_{i,\ldots k}$.

(ii) A term with a *simultaneous substitution* (or just *substitution*) postfix $[y_1 := Q_{1,\ldots k}]$ denotes the term obtained by replacing free occurrences of the variables $y_1, \ldots, y_k$ by the corresponding 'substituend' terms $Q_1, \ldots, Q_k$:

$$x\,[y_1 := Q_{1,\ldots k}] \equiv x \qquad \text{if } x \neq y_1, \ldots, y_k \tag{1}$$

$$y_1\,[y_1 := Q_{1,\ldots k}] \equiv Q_1 \tag{2}$$

$$(\lambda x.M)\,[y_1 := Q_{1,\ldots k}] \equiv \lambda x.\, M[y_1 := Q_{1,\ldots k}] \tag{3}$$

$$(MN)\,[y_1 := Q_{1,\ldots k}] \equiv M[y_1 := Q_{1,\ldots k}]\, N[y_1 := Q_{1,\ldots k}] \tag{4}$$

where it suffices in (2) to compare against the first variable in the binder set since they are unordered.

Remark. The variable convention 2.4 implies the usual $x \neq y_1, \ldots, y_k$ and $x \notin \mathrm{FV}(Q_1 \ldots Q_k)$ in (3). We could satisfy this explicitly by writing

$$(\lambda x.M)\,[y_1 := Q_{1,\ldots k}] \equiv \lambda x'.\, M[x := x'][y_1 := Q_{1,\ldots k}]$$

requiring instead that the introduced x' be "fresh," *i.e.*, $x' \notin \mathrm{FV}(M\, Q_1 \ldots Q_k)$. There is not much point in doing it this way, though, since it clearly does not change the opaque λ-term in question, as can easily be checked by comparing to (!) of definition 2.3(iv).

2.6 Definition. The semantics of the λ-calculus is given by the *β-reduction step* rule

$$(\lambda x.M)N \Rightarrow M[x := N] \tag{β}$$

and the requirement that $\Rightarrow$ is a reduction step. The theory containing just (β) is denoted $\boldsymbol{\lambda}$: we write $\boldsymbol{\lambda} \vdash M \Rightarrow N$ when the reduction $M \Rightarrow N$ is provable using (β) on Λ.

Remark. As any rewrite rule with variables the β-rule defines an infinite number of 'ground term rewrite rules.' Notice, however, that it does not define a regular CTRS because the right hand side (*RHS*) of each simple rewrite rule depends on the actual subterms M and N of the redex matching the left hand side (*LHS*).

This concludes the summary of the λ-calculus. Finally we recall some properties of it; proofs of these are standard in the literature.

2.7 Propositions.

(i) *Substitution lemma:* $M \Rightarrow\!\!\!\!\Rightarrow P$ *and* $N \Rightarrow\!\!\!\!\Rightarrow Q$ *implies* $(\lambda x.M)N \Rightarrow\!\!\!\!\Rightarrow P[x := Q]$.

(ii) *If* $x \notin \mathrm{FV}(M)$ *then* $M[x := N] \equiv M$.

2.8 Theorem. $\boldsymbol{\lambda}$ *is confluent.*

We will also be referring to λ-models as explained in [4, §5.2] but this is not essential to the presentation. We only mention the following:

2.9 Definition. A theory $\boldsymbol{\pi}$ over terms Π is a (non-trivial) λ-*model*, notation $\boldsymbol{\pi} \models \boldsymbol{\lambda}$, iff there is an interpretation $[\![\_]\!] \colon \Pi \to \Lambda$ that defines a *simulation*, *i.e.*, $\boldsymbol{\pi} \vdash P \Leftrightarrow Q$ implies $\boldsymbol{\lambda} \vdash [\![P]\!] \Leftrightarrow [\![Q]\!]$ which is *weakly extensional*, *i.e.*, for all $x \in \mathbb{V}$ and $M, N \in \Lambda$, $\boldsymbol{\lambda} \vdash M \Leftrightarrow N$ implies $\boldsymbol{\lambda} \vdash \lambda x.[\![M]\!] \Leftrightarrow \lambda x.[\![N]\!]$.

3 "let" is Explicit Substitution

We will now concentrate on the interpretation of **let** and acyclic substitution. We show how the mechanics of substitution may be "build in" (or "delayed") using a regular CTRS by presenting an alternate formulation of the untyped $\lambda\sigma$-calculus of Abadi, Cardelli, Curien, and Lévy [1] using names (instead of de Bruijn "nameless dummies" [6]). The calculus is thus not restricted to closed terms, and the similarity with **let** is maintained.

We first motivate and define our $\lambda\sigma$-calculus. Then we investigate the 'σ-calculus' of just the rewrite rules for eliminating explicit acyclic substitutions, and show that this is strongly normalising, confluent, and simulates **let**. Based on this we show how the $\lambda\sigma$-calculus may simulate the λ-calculus and thus that it is a λ-model.

$\lambda\sigma$-calculus

As discussed above, the intuitive denotation of **let** is a standard λ-term with the same meaning as an acyclic substitution, *i.e.*,

$$\textbf{let } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } M \quad \approx \quad M[x_1 := N_1, \ldots_k]$$

This motivates the following definition of our $\lambda\sigma$-calculus which contains λ-terms decorated with "explicit simultaneous acyclic substitution" components designated by $\langle\rangle$s.

3.1 Definition. Assume a set $\mathbb{V}$ of variables (ranged over by $x, y, z, \ldots$). The $\lambda\sigma$-*calculus* is defined as follows:

(i) The $\lambda\sigma$-*terms* are defined inductively by

$$\begin{array}{rcl} S & ::= & M\,\sigma \\ M & ::= & x \mid (\lambda x.S) \mid (ST) \\ \sigma & ::= & \langle x_1 := S_1, \ldots_k \rangle \end{array}$$

using the binder set notation of definition 2.5(i); S, T, U, ..., will range over $\lambda\sigma$-terms, M, N, P, Q, ..., over the "λ-components" stripped of the substitution, and σ, ρ, π, γ, ..., over explicit substitutions.

(ii) A *bound variable* x is x when occuring inside M of $\lambda x.M$ or $M\langle x := \cdots\rangle$. The *bound variable set* of a $\lambda\sigma$-term is defined correspondingly by

$$\begin{aligned}
\mathrm{BV}(x) &= \varnothing \\
\mathrm{BV}(\lambda x.S) &= \{x\} \cup \mathrm{BV}(S) \\
\mathrm{BV}(ST) &= \mathrm{BV}(S) \cup \mathrm{BV}(T) \\
\mathrm{BV}(M\langle x_1 := S_1, \ldots_k\rangle) &= \mathrm{BV}(M) \cup \{x_1, \ldots, x_k\} \cup \mathrm{BV}(S_1 \ldots S_k)
\end{aligned}$$

We will refer to the 'bound variables of substitutions' using the convention $\mathrm{BV}(\sigma) = \mathrm{BV}(x\sigma)$.

(iii) The *free variable set* of a $\lambda\sigma$-term contains those variables not bound; inductively:

$$\begin{aligned}
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(\lambda x.S) &= \mathrm{FV}(S) \setminus \{x\} \\
\mathrm{FV}(ST) &= \mathrm{FV}(S) \cup \mathrm{FV}(T) \\
\mathrm{FV}(M\langle x_1 := S_1, \ldots_k\rangle) &= (\mathrm{FV}(M) \setminus \{x_1, \ldots, x_k\}) \cup \mathrm{FV}(S_1 \ldots S_k)
\end{aligned}$$

We will also refer to the 'free variables of substitutions' using the convention $\mathrm{FV}(\sigma) = \mathrm{FV}((\lambda x.x)\sigma)$.

(iv) A term with a *renaming* postfix $[x := y]$ denotes the term obtained by changing 'free occurrences' of x in it to y, inductively specified by:

$$\begin{aligned}
x\,[x := y] &\equiv y \\
z\,[x := y] &\equiv z && \text{if } z \neq x \\
(\lambda z.S)\,[x := y] &\equiv \lambda z'.\, S[z := z'][x := y] && \text{with } z' \notin \mathrm{FV}(x\,y\,S) \\
(ST)\,[x := y] &\equiv S[x := y]\,T[x := y] \\
M\langle z_1 := S_1, \ldots_k\rangle\,[x := y] & \\
&\equiv M[z_1 := z'_1] \ldots [z_k := z'_k][x := y]\,\langle z'_1 := S_1[x := y], \ldots_k\rangle \\
& && \text{with } x, y \neq z'_1, \ldots, z'_k
\end{aligned}$$

(v) *Equivalence* of $\lambda\sigma$-terms means "syntactic equality modulo renaming and garbage collection," is denoted $\equiv$, and defined inductively by

$$\begin{aligned}
x &\equiv x \\
\lambda x.S &\equiv \lambda y.T && \text{if } S[x := y] \equiv T \\
ST &\equiv UV && \text{if } S \equiv U \wedge T \equiv V \\
M\langle x_1 := S_1, \ldots_k\rangle &\equiv N\langle y_1 := T_1, \ldots_k\rangle \\
& && \text{if } M[x_1 := y_1] \ldots [x_k := y_k] \equiv P \wedge \forall i\colon S_i \equiv T_i
\end{aligned}$$

We will also relax this notation and write $\sigma \equiv \rho$ meaning $M\sigma \equiv M\rho$ for all M.

(vi) $\Lambda\sigma$ is the set of all *opaque* $\lambda\sigma$*-terms*: $\Lambda\sigma = \lambda\sigma\text{-terms}/\equiv$. Any concrete $\lambda\sigma$-term representing an object in $\Lambda\sigma$ is called a *representative* of that term.

Remarks. Some interesting things to note are the following:

(i) Variables may now be either 'λ-bound' or 'σ-bound,' *i.e.*, bound by an enclosing abstraction or explicit substitution. The variable convention as formulated in 2.4 is unchanged, however.

(ii) All the operations make sense on individual substitutions even though substitutions are not separate syntactic entities but always 'glued onto' some $\lambda\sigma$-term.

We are now able to extract the 'bookkeeping' performed during substitution from β-reduction. The following defines $\lambda\sigma$-reduction using the idea that the substitutions to be performed on a term should be that attached to the term and then those of all superterms (*i.e.*, terms containing it as a subterm). Thus we will not define implicit substitution on $\lambda\sigma$-terms—hence renaming is not a special case of substitution in the explicit calculus.

First we will establish that we can use the variable convention.

3.2 Proposition. *Any $\lambda\sigma$-term has representatives satisfying the variable convention 2.4.*

Proof. Easy induction over the depth of the terms of $\Lambda\sigma$: assume it holds for all subterms of $Q \equiv M\langle x_1 := S_1, \ldots_k\rangle$, *i.e.*, for M, S_1, ... , S_k. This has a representative $M[x_1 := x_1'] \ldots [x_k := x_k']\langle x_1' := S_1, \ldots_k\rangle$ where we have chosen x_1', ... , x_k' not in $\mathrm{BV}(M) \cup \mathrm{BV}(S_1) \cup \cdots \cup \mathrm{BV}(S_k) \cup \mathrm{FV}(Q)$. Then $\mathrm{BV}(Q)$ is disjoint from $\mathrm{FV}(Q)$ as required. ■

This makes the following definition of reduction much simpler since variable capture does not become a problem. The primary idea is the encoding of the notion of a *scope nesting* in the $*$ operation.

3.3 Definition. The $\lambda\sigma$*-reduction steps* are syntactic β-reduction (from 2.6) and explicit rules for substitution (from 2.5):

$$(\beta\sigma) \qquad ((\lambda x.M\sigma)\rho\, S)\gamma \Rightarrow M\,(\sigma * \rho * \langle x := S\rangle * \gamma)$$

$$(\sigma_1) \qquad x\,\langle y_1 := S_1, y_2 := S_2, \ldots_k\rangle \Rightarrow x\,\langle y_2 := S_2, \ldots_k\rangle \qquad \text{if } x \not\equiv y_1$$

$$(\sigma_2) \qquad y_1\,\langle y_1 := S_1, \ldots_k\rangle \Rightarrow S_1$$

$$(\sigma_3) \qquad (\lambda x.M\sigma)\rho \Rightarrow (\lambda x.\, M\,(\sigma * \rho))\,\langle\rangle \qquad \text{if } \rho \not\equiv \langle\rangle$$

$$(\sigma_4) \qquad (M\sigma\, N\pi)\rho \Rightarrow (M(\sigma * \rho)\, N(\pi * \rho))\,\langle\rangle \qquad \text{if } \rho \not\equiv \langle\rangle$$

where the *substitution nesting* operator, $*$, is defined as follows for explicit substitutions $\sigma = \langle x_1 := M_1\sigma_1, \ldots_j\rangle$ and $\rho = \langle y_1 := N_1\rho_1, \ldots_k\rangle$:

$$\sigma * \rho \equiv \langle x_1 := M_1(\sigma_1 * \rho), \ldots_j, y_1 := N_1\rho_1, \ldots_k\rangle$$

The theory $\boldsymbol{\sigma}$ contains the least stable and compatible $\Rightarrow$ that includes the σ-rules $(\sigma_1 \ldots \sigma_4)$; $\boldsymbol{\lambda\sigma}$ similarly for $(\beta\sigma) + \boldsymbol{\sigma}$.

Remarks.

(i) The side conditions of (σ_3, σ_4) ensure that $\Rightarrow$ is not reflexive.
(ii) The only σ-rule that differs essentially from those of definition 2.3 is (σ_1) where the binders are disposed of one by one.
(iii) The formulation as a CTRS means that there is no need to prove substitutivity since that is inherent in the fact that the relation is the 'stable compatible closure' of the rewrite rules (*cf.* [10, §3.2]).
(iv) The variable convention 2.4 ensures both that *variable capture* does not occur and that $*$ is always defined since the variables of the combined substitutions must be distinct in the rewritten representatives, *e.g.*, variable capture of x is avoided in $(\beta\sigma)$ because it ensures $x \notin \mathrm{FV}(\rho)$—this can be seen as follows: assume $x \in \mathrm{FV}(\rho)$. Then either x is free in the entire term, which is forbidden, or x is both λ- and σ-bound, which is equally forbidden.

3.4 Example. The **let** example in the introduction would thus be coded as the $\lambda\sigma$-term $((a+b)\times(a-b))\ \langle a := 2^3, b := 3^2\rangle$ assuming suitable definitions of arithmetic was provided.

The nesting operator has nice properties; we will make intensive use of this in the proofs below.

3.5 Proposition. *Nesting is associative:* $(\sigma * \rho) * \gamma \equiv \sigma * (\rho * \gamma)$.

Proofs. Assume $\forall i\colon \sigma_i = \langle x_{i1} := M_{i1}\sigma_{11}, \ldots in_i\rangle$:

$$\begin{aligned}
&(\langle x_1 := M_1\sigma_1, \ldots i\rangle * \langle y_1 := N_1\rho_1, \ldots j\rangle) * \langle z_1 := P_1\pi_1, \ldots k\rangle \\
&\qquad \equiv \langle x_1 := M_1\langle x_{i1} := M_{i1}(\sigma_{11} * \langle y_1 := N_1\rho_1, \ldots j\rangle), \ldots in_i\,, \\
&\qquad\qquad y_1 := N_1(\rho_1 * \langle z_1 := P_1\pi_1, \ldots k\rangle), \ldots j\,, \\
&\qquad\qquad z_1 := P_1\pi_1, \ldots k\rangle, \ldots i\,, \\
&\qquad\quad y_1 := N_1(\rho_1 * \langle z_1 := P_1\pi_1, \ldots k\rangle), \ldots j\,, \\
&\qquad\quad z_1 := P_1\pi_1, \ldots k\rangle \\
&\qquad \equiv \langle x_1 := M_1\sigma_1, \ldots i\rangle * (\langle y_1 := N_1\rho_1, \ldots j\rangle * \langle z_1 := P_1\pi_1, \ldots k\rangle) \qquad \blacksquare
\end{aligned}$$

$\boldsymbol{\sigma}$ *and substitution*

With this we are ready to express how to simulate substitution using explicit substitution—or rather: to justify formally why the $\boldsymbol{\sigma}$-rules are called "explicit substitution" in the first place. We first establish the confluency of the explicit substitution CTRS, then we formulate a simulation and prove it correct.

3.6 Propositions.

(i) $\boldsymbol{\sigma}$ *is strongly normalising.*
(ii) $\boldsymbol{\sigma}$ *is locally confluent.*
(iii) $\boldsymbol{\sigma}$ *is confluent.*

Proofs.

(i) σ is strongly normalising for any $\lambda\sigma$-term by induction over the depth of $\lambda\sigma$-terms since all the reduction steps either remove bindings or push bindings "inwards"

replacing outer substitutions by $\langle\rangle$ that can never be the substitution of a redex. ✓

(ii) To prove local confluence we only have to consider the *critical pairs* (*cf.* [12] and [10, §3.2]) of reductions where an inner redex 'covers' more that a single symbol of the outer redex; all other cases follow by compatibility of the reduction. So for each possible pair M, N of redices where the second is contained in more than one variable of the LHS of the reduction for the first we prove $M \Downarrow N$. In each case the left reduction reduces the entire top term and the right the underlined redex of it. We get the following cases:

Case (σ_3) containing (σ_1): Assume $x \neq y_1$ and $\rho \not\equiv \langle\rangle$.

$$\begin{array}{ccc} & (\lambda x.\,\underline{x\langle y_1 := Q_1\sigma_1, y_2 := Q_2\sigma_2, \ldots_k\rangle})\,\rho & \\ {\scriptstyle\sigma_3}\swarrow & & \searrow{\scriptstyle\sigma_1} \\ (\lambda x.x\langle y_1 := Q_1\sigma_1, y_2 := Q_2\sigma_2, \ldots_k\rangle * \rho)\langle\rangle & & (\lambda x.x\langle y_2 := Q_2\sigma_2, \ldots_k\rangle)\,\rho \\ {\scriptstyle\sigma_1}\searrow & & \swarrow{\scriptstyle\sigma_3} \\ & (\lambda x.x\langle y_2 := Q_2\sigma_2, \ldots_k\rangle * \rho)\langle\rangle \quad ✓ & \end{array}$$

Case (σ_3) containing (σ_2): Assume $\rho \not\equiv \langle\rangle$.

$$\begin{array}{ccc} & (\lambda x.\,\underline{y_1\langle y_1 := Q_1\sigma_1, \ldots_k\rangle})\,\rho & \\ {\scriptstyle\sigma_3}\swarrow & & \searrow{\scriptstyle\sigma_2} \\ (\lambda x.y_1\langle y_1 := Q_1\,(\sigma_1 * \rho), \ldots_k\rangle)\langle\rangle & & (\lambda x.Q_1\sigma_1)\rho \\ {\scriptstyle\sigma_2}\searrow & & \swarrow{\scriptstyle\sigma_3} \\ & (\lambda x.Q_1\,(\sigma_1 * \rho))\langle\rangle \quad ✓ & \end{array}$$

Case (σ_3) containing (σ_3): Assume $\sigma, \rho \not\equiv \langle\rangle$.

$$\begin{array}{ccc} & (\lambda x.\,\underline{(\lambda y.M\sigma)\rho})\,\gamma & \\ & {\scriptstyle\sigma_3}\searrow & \\ {\scriptstyle\sigma_3}\swarrow & & (\lambda x.(\lambda y.M(\sigma * \rho))\langle\rangle)\,\gamma \\ (\lambda x.(\lambda y.M\sigma)\,(\rho * \gamma))\,\langle\rangle & & {\scriptstyle\sigma_3}\Downarrow \\ {\scriptstyle\sigma_3}\searrow & & (\lambda x.(\lambda y.M(\sigma * \rho))\,(\langle\rangle * \gamma))\,\langle\rangle \\ & {\scriptstyle\sigma_3}\swarrow & \\ & (\lambda x.(\lambda y.M\,(\sigma * \rho * \gamma))\,\langle\rangle)\,\langle\rangle \quad ✓ & \end{array}$$

Case (σ_3) containing (σ_4): Assume $\sigma, \rho \not\equiv \langle\rangle$.

$$\begin{array}{ccc} & (\lambda x.\,\underline{(M\sigma\,N\rho)\pi})\,\gamma & \\ & {\scriptstyle\sigma_4}\searrow & \\ {\scriptstyle\sigma_3}\swarrow & & (\lambda x.(M(\sigma * \pi)\,N(\rho * \pi))\,\langle\rangle)\,\gamma \\ (\lambda x.\,(M\sigma\,N\rho)\,(\pi * \gamma))\,\langle\rangle & & {\scriptstyle\sigma_3}\Downarrow \\ {\scriptstyle\sigma_4}\searrow & & (\lambda x.(M(\sigma * \pi)\,N(\rho * \pi))\,\gamma)\,\langle\rangle \\ & {\scriptstyle\sigma_4}\swarrow & \\ & (\lambda x.(M(\sigma * \pi * \gamma)\,N(\rho * \pi * \gamma))\,\langle\rangle)\,\langle\rangle \quad ✓ & \end{array}$$

Case (σ_4) containing (σ_1): Assume $x \neq y_1$ and $\gamma \neq \langle\rangle$.

$$\begin{array}{ccc} & (\,\underline{x\langle y_1 := S_1, y_2 := S_2, \ldots_k\rangle}\;N\rho)\gamma & \\ {\scriptstyle\sigma_4}\swarrow & & \searrow{\scriptstyle\sigma_1} \\ (x(\langle y_1 := S_1, y_2 := S_2, \ldots_k\rangle * \gamma)\,N(\rho * \gamma))\,\langle\rangle & & (x\langle y_2 := S_2, \ldots_k\rangle\;N\rho)\gamma \\ {\scriptstyle\sigma_1}\searrow & & \swarrow{\scriptstyle\sigma_4} \\ & (x(\langle y_2 := S_2, \ldots_k\rangle * \gamma)\;N(\rho * \gamma))\,\langle\rangle \quad ✓ & \end{array}$$

Case (σ_4) containing (σ_2): Assume $\gamma \neq \langle\rangle$.

$$\begin{array}{ccc} & (\underline{y_1\langle y_1 := M_1\sigma_1,\ldots_k\rangle}\ N\rho)\gamma & \\ {\scriptstyle\sigma_4}\swarrow & & \searrow{\scriptstyle\sigma_2} \\ (x(\langle y_1 := M_1\sigma_1,\ldots_k\rangle * \gamma)\ N(\rho*\gamma))\langle\rangle & & (M_1\sigma_1\ N\rho)\gamma \\ {\scriptstyle\sigma_2}\searrow & & \swarrow{\scriptstyle\sigma_4} \\ & ((M_1(\sigma_1*\gamma)\ N(\rho*\gamma))\langle\rangle \quad \checkmark & \end{array}$$

Case (σ_4) containing (σ_3): Assume $\pi, \gamma \neq \langle\rangle$.

$$\begin{array}{ccc} & (\underline{(\lambda x.M\sigma)\pi}\ N\rho)\gamma & \\ {\scriptstyle\sigma_4}\swarrow & & \searrow{\scriptstyle\sigma_3} \\ ((\lambda x.M\sigma)(\pi*\gamma)\ N(\rho*\gamma))\langle\rangle & & ((\lambda x.M(\sigma*\pi))\langle\rangle\ N\rho)\gamma \\ {\scriptstyle\sigma_3}\searrow & & \downarrow{\scriptstyle\sigma_4} \\ & ((\lambda x.M(\sigma*\pi*\gamma))\langle\rangle N(\rho*\gamma))\langle\rangle \quad \checkmark & ((\lambda x.M(\sigma*\pi))\gamma\ N(\rho*\gamma))\langle\rangle \\ & & \swarrow{\scriptstyle\sigma_3} \end{array}$$

Case (σ_4) containing (σ_4): Assume $\sigma, \gamma \neq \langle\rangle$.

$$\begin{array}{ccc} & (\underline{(M\pi P\pi')\sigma}\ N\rho)\gamma & \\ {\scriptstyle\sigma_4}\swarrow & & \searrow{\scriptstyle\sigma_4} \\ ((M\pi P\pi')(\sigma*\gamma)\,N(\rho*\gamma))\langle\rangle & & ((M(\pi*\sigma)\ P(\pi'*\sigma))\langle\rangle\,N\rho)\gamma \\ {\scriptstyle\sigma_4}\searrow & & \swarrow{\scriptstyle\sigma_4} \\ & ((M(\pi*\sigma*\gamma)\,P(\pi'*\sigma*\gamma))\langle\rangle\,N(\rho*\gamma))\langle\rangle \quad \checkmark & \end{array}$$

repeating the last four cases symmetrically for applications where the right subterm is reduced. We have used the definition and associativity of $*$ almost everywhere. ✓

(iii) Follows from the previous two propositions by 2.2(i). ■

Next we present the promised translations of λ-terms to $\lambda\sigma$-terms and back.

3.7 Definition.

(i) *Lifting* of λ-terms to $\lambda\sigma$-terms, $\ulcorner\_\urcorner$: $\Lambda \to \Lambda\sigma$, is defined to add dummy substitutions by

$$\begin{aligned} \ulcorner x \urcorner &= x\,\langle\rangle \\ \ulcorner \lambda x.M \urcorner &= (\lambda x.\ulcorner M\urcorner)\,\langle\rangle \\ \ulcorner MN \urcorner &= (\ulcorner M\urcorner\,\ulcorner N\urcorner)\,\langle\rangle \end{aligned}$$

We will write $\ulcorner M\urcorner\sigma$ for $M'\sigma$ where $M'\langle\rangle = \ulcorner M\urcorner$.

(ii) *Sinking* of $\lambda\sigma$-terms to λ-terms, $\llcorner\_\lrcorner$: $\Lambda\sigma \to \Lambda$, is defined to 'expand' all substitutions by

$$\begin{aligned} \llcorner x\langle y_1 := S_1,\ldots_k\rangle \lrcorner &= x \\ \llcorner y_1\langle y_1 := S_1,\ldots_k\rangle \lrcorner &= \llcorner S_1 \lrcorner \\ \llcorner(\lambda x.S)\,\langle y_1 := S_1,\ldots_k\rangle \lrcorner &= (\lambda x.\llcorner S\lrcorner)\,[y_1 := \llcorner S_1\lrcorner,\ldots_k] \\ \llcorner(MN)\,\langle y_1 := S_1,\ldots_k\rangle \lrcorner &= (\llcorner M\lrcorner\,\llcorner N\lrcorner)\,[y_1 := \llcorner S_1\lrcorner,\ldots_k] \end{aligned}$$

This suggests the following simulation.

3.8 Lemma. *$\boldsymbol{\sigma}$ simulates substitution: If $M, N_1, \ldots \in \Lambda$ then*

$$\boldsymbol{\sigma} \vdash \ulcorner M \urcorner \langle x_1 := \ulcorner N_1 \urcorner, \ldots_k \rangle \twoheadrightarrow \ulcorner M[x_1 := N_1, \ldots_k] \urcorner$$

Proof. Given that the reduction is strongly normalising and confluent we just have to choose a reduction order that in fact simulates substitution We decide to always reduce the innermost non-empty substitution first, choosing (σ_2) over (σ_1) when possible. Then the σ-rules may be expressed as

$$\begin{aligned}
x\,\langle y_1 := N_1\langle\rangle, y_2 := N_2\langle\rangle, \ldots_k \rangle &\twoheadrightarrow x\,\langle\rangle && \text{if } x \neq y_1, \ldots, y_k \\
y_1\,\langle y_1 := N_1\langle\rangle, \ldots_k \rangle &\Rightarrow N_1\langle\rangle \\
(\lambda x.M\langle\rangle)\,\rho &\Rightarrow (\lambda x.M\rho)\,\langle\rangle && \text{if } \rho \not\equiv \langle\rangle \\
(M\langle\rangle\ N\langle\rangle)\,\rho &\Rightarrow (M\rho\ N\rho)\,\langle\rangle && \text{if } \rho \not\equiv \langle\rangle
\end{aligned}$$

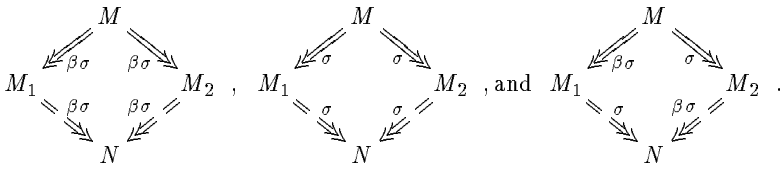
(where the first rule is exactly k (σ_1) reduction steps). This may be seen to be the same as $(1 \ldots 4)$ of 2.5 by applying $\llcorner - \lrcorner$ of definition 3.7(ii). By induction over the depth of terms it may be deduced that the $\boldsymbol{\sigma}$-normal form reached is the required lifting of the term after substitution. ■

$\boldsymbol{\lambda\sigma}$ *may simulate* $\boldsymbol{\lambda}$

Finally we are able to prove that $\boldsymbol{\lambda\sigma}$ is a λ-model by giving the complete simulation of $\boldsymbol{\lambda}$-reduction in $\boldsymbol{\lambda\sigma}$. We first establish that $\boldsymbol{\lambda\sigma}$ is confluent and then give the simulation theorem.

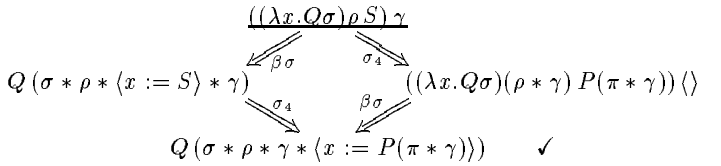
3.9 Lemma. *$\boldsymbol{\lambda\sigma}$ is confluent.*

Proof. By the Hindley-Rosen lemma 2.2(ii) it suffices to show



The leftmost diagram is trivially true because $(\beta\sigma)$ rule commutes with itself by the compatibility of it (it has no critical pairs with itself). The middle diagram was proven as proposition 3.6(iii). This leaves the rightmost diagram. Since $(\beta\sigma)$ alone is obviously strongly normalising (each removes a redex without creating new ones or copying any) as is $\boldsymbol{\sigma}$ by proposition 3.6(i), it is is sufficient to show that they commute as we may then conclude the leftmost diagram by induction. Again this is done by checking critical pairs where a $(\beta\sigma)$- and a $\boldsymbol{\sigma}$-redex overlap:

Case Shared $(\beta\sigma)$ and (σ_4) redex:



Case $(\beta\sigma)$ containing (σ_1):

$$\begin{array}{ccc} & ((\lambda x.\, \underline{z\langle y_1 := S_1, y_2 := S_2, \ldots_k\rangle})\rho\, T)\,\gamma & \\ {\scriptstyle\beta\sigma}\swarrow\swarrow & & \searrow\searrow{\scriptstyle\sigma_1} \\ z\,((\langle y_1 := S_1, y_2 := S_2, \ldots_k\rangle * \rho) * \langle x := T\rangle * \gamma) & & ((\lambda x.z\langle y_2 := S_2, \ldots_k\rangle)\rho\, T)\,\gamma \\ {\scriptstyle\sigma_1}\searrow\searrow & & \swarrow\swarrow{\scriptstyle\beta\sigma} \\ & z\,(\langle y_2 := S_2, \ldots_k\rangle * \rho * \langle x := T\rangle * \gamma) & \end{array}$$

Case $(\beta\sigma)$ containing (σ_2):

$$\begin{array}{ccc} & ((\lambda x.\, \underline{y_1\langle y_1 := M_1\sigma_1, \ldots_k\rangle})\rho\, T)\,\gamma & \\ {\scriptstyle\beta\sigma}\swarrow\swarrow & & \searrow\searrow{\scriptstyle\sigma_2} \\ y_1\,((\langle y_1 := M_1\sigma_1, \ldots_k\rangle * \rho) * \langle x := T\rangle * \gamma) & & ((\lambda x.M_1\sigma_1)\rho\, T)\,\gamma \\ {\scriptstyle\sigma_2}\searrow\searrow & & \swarrow\swarrow{\scriptstyle\beta\sigma} \\ & M_1\,(\sigma_1 * \rho * \langle x := T\rangle * \gamma) \qquad \checkmark & \end{array}$$

Case $(\beta\sigma)$ containing (σ_3):

$$\begin{array}{ccc} & (\underline{(\lambda x.Q\sigma)\rho}\ S)\,\gamma & \\ & \Big\| {\scriptstyle\beta\sigma} \quad \searrow\searrow{\scriptstyle\sigma_3} & \\ & \swarrow\swarrow{\scriptstyle\beta\sigma} \quad ((\lambda x.\, Q(\sigma * \rho))\langle\rangle\ S)\,\gamma & \\ Q\,((\sigma * \rho) * \langle x := S\rangle * \gamma) \qquad \checkmark & & \end{array}$$

Case (σ_3) containing $(\beta\sigma)$: For $y \notin \{x\} \cup \mathrm{BV}(\sigma) \cup \mathrm{BV}(\rho)$ then

$$\begin{array}{ccc} & (\lambda y.\, \underline{((\lambda x.Q\sigma)\rho\ S)\,\gamma}\,)\theta & \\ {\scriptstyle\sigma_3}\swarrow\swarrow & & \searrow\searrow{\scriptstyle\beta\sigma} \\ (\lambda y.((\lambda x.Q\sigma)\rho\ S)\,(\gamma * \theta))\,\langle\rangle & & (\lambda y.Q\,(\sigma * \rho * \langle x := S\rangle * \gamma))\,\theta \\ {\scriptstyle\beta\sigma}\searrow\searrow & & \swarrow\swarrow{\scriptstyle\sigma_3} \\ & (\lambda y.Q\,(\sigma * \rho * \langle x := S\rangle * \gamma * \theta))\,\langle\rangle \qquad \checkmark & \end{array}$$

Case (σ_4) containing $(\beta\sigma)$: Here is just the case where the $(\beta\sigma)$ redex is the left subtree of the (σ_4) redex; the symmetric situation is similar.

$$\begin{array}{ccc} & (\underline{((\lambda x.Q\sigma)\rho\ S)\,\gamma}\ N\pi)\theta & \\ {\scriptstyle\sigma_4}\swarrow\swarrow & & \searrow\searrow{\scriptstyle\beta\sigma} \\ (((\lambda x.Q\sigma)\rho\ S)\,(\gamma * \theta)\,N(\pi * \theta))\,\langle\rangle & & ((Q\,(\sigma * \rho * \langle x := S\rangle * \gamma))\,N\pi)\theta \\ {\scriptstyle\beta\sigma}\searrow\searrow & & \swarrow\swarrow{\scriptstyle\sigma_4} \\ & (Q(\sigma * \rho * \langle x := S\rangle * \gamma * \theta)\,N(\pi * \theta))\,\langle\rangle \qquad \checkmark & \end{array}$$

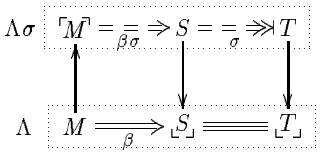
We have exploited the associativity of $*$ freely. ■

Remark. In [1, §3.2] the authors remark that using names (instead of de Bruijn indices) in their explicit substitution calculus destroys the confluency properties. The conflict of their system does not arise here because we have kept the variable convention: variables are semantic rather than syntactic which means that it is not possible to "rename into a corner," *i.e.*, to do some renaming that can't be undone, destroying confluency.

With this we may state the main result of this section.

3.10 Theorem. $\boldsymbol{\lambda\sigma} \models \boldsymbol{\lambda}$.

Proof. The weak extensionality is a direct consequence of the compatibility of $\boldsymbol{\lambda\sigma}$. Since we already have that $\boldsymbol{\lambda\sigma}$ is confluent from lemma 3.9 then simulation follows from completion of the diagram



where the $\downarrow$- and $\uparrow$-arrows indicate lifting and sinking of definition 3.7 as indicated. But we have already shown this: the top left reduction is $(\beta\sigma)$ of definition 3.3 and the top right one is lemma 3.8 on the resulting $\lambda\sigma$-term that must contain exactly one non-$\langle\rangle$ substitution, namely the one produced by $(\beta\sigma)$. ■

Summary

We have shown how $\boldsymbol{\lambda\sigma}$ provides an explicit operational model for the λ-calculus enriched with **let** as used in functional programming languages using the identification

$$\ulcorner \textbf{let } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } M \urcorner \quad \equiv \quad \ulcorner M \urcorner \langle x_1 := \ulcorner N_1 \urcorner, \ldots_k \rangle$$

because by lemma 3.8 and theorem 3.10 we have

$$\begin{aligned} \boldsymbol{\lambda\sigma} \quad \vdash \quad & \ulcorner (\lambda x.M) N \urcorner \quad \Rightarrow \quad \ulcorner M[x := N] \urcorner \\ & \ulcorner \textbf{let } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } M \urcorner \quad \Rightarrow \quad \ulcorner M[x_1 := N_1, \ldots_k] \urcorner \end{aligned}$$

4 "letrec" is Explicit Cyclic Substitution

This section will generalise the idea of the previous section to handle local *recursive* definitions as required by the **letrec** construction discussed in the introduction. This leads to a modification of the $\lambda\sigma$-calculus where the explicit substitution is *cyclic*—this is named the $\lambda\mu$-calculus due to its similarity with fixed point induction operator in, *e.g.*, Park's μ-calculus [16].

We start out with a motivation for and definition of the $\lambda\mu$-calculus. We then follow the same strategy as in the previous section: We first study the 'μ-calculus' part of the theory, prove it confluent, and show that it provides an operational interpretation of **letrec**. Then we show that the combination with the β-reduction step is also confluent, and finally that the $\lambda\mu$-calculus is a model of the $\lambda\sigma$-calculus (and thus also a λ-model).

$\lambda\mu$-calculus

Our starting point is the intuition of the introduction that "recursion is just cyclic definition." In terms of substitutions this corresponds to *unfolding* as

captured by the following equation for **letrec** terms:

$$\begin{aligned}&\textbf{letrec } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } M\\ &\quad\approx M[\ x_1 := \textbf{letrec } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } N_1\ ,\\ &\qquad\vdots\\ &\qquad x_k := \textbf{letrec } x_1 = N_1 \textbf{ and} \ldots \textbf{and } x_k = N_k \textbf{ in } N_k\ ]\end{aligned}$$

With this $Y \equiv \lambda f.\ \textbf{letrec } y = fy \textbf{ in } y$ is a "cyclic fixed point combinator" because

$$YM \ \approx\ \textbf{letrec } y = My \textbf{ in } y \ \approx\ M\,(\textbf{letrec } y = My \textbf{ in } y) \ \approx\ M(YM)$$

There is one problem with this definition that must be handled: **letrec** bindings do not disappear when unused—rather they are "sticky" and thus we must ensure that there is a way to "garbage collect" unused bindings. This motivates the invention of the "explicit simultaneous cyclic substitution" calculus.

4.1 Definition. Assume a set $\mathbb{V}$ of variables (ranged over by $x, y, z, \ldots$). The *$\lambda\mu$-calculus* is defined as follows (where details that differ more than just syntactically from those of definition 3.1 are tagged with !s):

(i) The *$\lambda\mu$-terms* are defined inductively by

$$\begin{aligned} S &::= M\,\mu \\ M &::= x \ \mid\ (\lambda x.S) \ \mid\ (ST) \\ \mu &::= \langle\!\langle x_1 := S_1, \ldots_k \rangle\!\rangle \end{aligned}$$

using the binder set notation of definition 2.5(i); S, T, U, ..., will range over $\lambda\mu$-terms, M, N, P, Q, ..., over the "λ-component" stripped of the substitution, and μ, ρ, π, γ, ..., over explicit cyclic substitutions.

(ii) The *defined variables* of a substitution is the set of the variables bound by it: $\mathrm{DV}(\langle\!\langle x_1 := S_1, \ldots_k \rangle\!\rangle) = \{x_1, \ldots, x_k\}$.

(iii) A *bound variable* x is x when occuring inside M or μ of $\lambda x.M\mu$, $M\langle\!\langle x := \ldots \rangle\!\rangle$, or $\ldots\langle\!\langle x := M\mu, \ldots \rangle\!\rangle$. The *bound variable set* of a $\lambda\mu$-term is defined correspondingly by

$$\begin{aligned} \mathrm{BV}(x) &= \varnothing \\ \mathrm{BV}(\lambda x.S) &= \{x\} \cup \mathrm{BV}(S) \\ \mathrm{BV}(ST) &= \mathrm{BV}(S) \cup \mathrm{BV}(T) \\ \mathrm{BV}(M\langle\!\langle x_1 := S_1, \ldots_k \rangle\!\rangle) &= \mathrm{BV}(M) \cup \{x_1, \ldots, x_k\} \cup \mathrm{BV}(S_1 \ldots S_k) \end{aligned}$$

We refer to the bound variables of cyclic substitutions using the convention $\mathrm{BV}(\mu) = \mathrm{BV}(x\mu)$.

(iv) The *free variable set* of a $\lambda\mu$-term is the set of variables not bound by an

enclosing abstraction or cyclic substitution; inductively:

$$\begin{aligned}
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(\lambda x.S) &= \mathrm{FV}(S) \setminus \{x\} \\
\mathrm{FV}(ST) &= \mathrm{FV}(S) \cup \mathrm{FV}(T) \\
\text{(!)} \qquad \mathrm{FV}(M\langle\!\langle x_1 := S_1, \ldots_k \rangle\!\rangle) &= \mathrm{FV}(M\, S_1 \ldots S_k) \setminus \{x_1, \ldots, x_k\}
\end{aligned}$$

We will also refer to the free variables of substitutions using the convention $\mathrm{FV}(\mu) = \mathrm{FV}((\lambda x.x)\mu)$.

(v) A term with a *renaming* postfix $[x := y]$ denotes the term obtained by changing 'free occurrences' of x in it to y, inductively specified

$$\begin{aligned}
x\,[x := y] &\equiv y \\
z\,[x := y] &\equiv z && \text{if } z \neq x \\
(\lambda z.S)\,[x := y] &\equiv \lambda z'.\, S[z := z'][x := y] && \text{with } z' \notin \mathrm{FV}(x\,y\,S) \\
(ST)\,[x := y] &\equiv S[x := y]\,T[x := y]
\end{aligned}$$

$$\begin{aligned}
\text{(!!)} \qquad & M\langle\!\langle z_1 := S_1, \ldots_k \rangle\!\rangle\,[x := y] \\
&\equiv M[z_1 := z'_1] \ldots [z_k := z'_k][x := y] \\
&\qquad \langle\!\langle z'_1 := S_1[z_1 := z'_1] \ldots [z_k := z'_k][x := y], \ldots_k \rangle\!\rangle \\
&\qquad\qquad \text{with } x, y \neq z'_1, \ldots, z'_k
\end{aligned}$$

(vi) *Equivalence* of $\lambda\mu$-terms means "syntactic equality modulo renaming and *garbage collection*," is denoted $\equiv$, and defined inductively by

$$\begin{aligned}
x &\equiv x \\
\lambda x.S &\equiv \lambda y.T && \text{if } S[x := y] \equiv T \\
ST &\equiv UV && \text{if } S \equiv U \wedge T \equiv V
\end{aligned}$$

$$\begin{aligned}
\text{(!!!)} \qquad & M\langle\!\langle x_1 := S_1, \ldots_j \rangle\!\rangle \equiv N\langle\!\langle y_1 := T_1, \ldots_k \rangle\!\rangle \\
&\qquad \text{if } j \geq k \wedge M[x_1 := y_1] \ldots [x_k := y_k] \equiv N \\
&\qquad\quad \wedge \forall i \leq k : S_i[x_1 := y_1] \ldots [x_k := y_k] \equiv T_i \\
&\qquad\quad \wedge \forall i > k : x_i \notin \mathrm{FV}(M\, S_1 \ldots S_k)
\end{aligned}$$

We will also relax this notation and write $\mu \equiv \rho$ meaning $M\mu \equiv M\rho$ for all M.

(vii) $\Lambda\mu$ is the set of all *opaque* $\lambda\mu$*-terms*: $\Lambda\mu = \lambda\mu$-terms$/\equiv$. Any concrete $\lambda\mu$-term representing an object in $\Lambda\mu$ is called a *representative* of that term.

Remarks. The main difference from $\lambda\sigma$ of definition 3.1 (apart from using μ as a metavariable instead of σ to avoid confusion) is of course that substitution variables are bound also in the substitution body. This results in the need for the changes above marked with !s:

(!) The free variables of a term should not include any substitution variables in the substituend terms.

(!!) When renaming substitution variables this must be done everywhere they are bound, *i.e.*, also in the substituend terms.

(!!!) Equivalence of substitution requires renaming the substitution variables inside the substituend term. Also garbage collection of unneeded binders is allowed without changing the opaque term in question—this was not needed for $\lambda\sigma$ since σ-binders may be removed one by one using (σ_1) whereas μ-binders do not disappear using (μ_1) unless the entire substitution may be disposed of.

4.2 Proposition. *Any $\lambda\mu$-term has representatives satisfying the variable convention 2.4.*

Proof. Induction over the depth of the terms of $\Lambda\mu$ very similar to that used to prove 3.2: assume it holds for all subterms of $Q \equiv M\langle\!\langle x_1 := S_1, \ldots_k \rangle\!\rangle$, *i.e.*, for M, S_1, ... , S_k. This has a representative $M[x_1 := x'_1]\ldots[x_k := x'_k]\langle\!\langle x'_1 := S_1[x_1 := x'_1]\ldots[x_k := x'_k], \ldots_k \rangle\!\rangle$ where we have chosen x'_1, ... , x'_k not in $\mathrm{BV}(M) \cup \mathrm{BV}(S_1) \cup \cdots \cup \mathrm{BV}(S_k) \cup \mathrm{FV}(Q)$. Then $\mathrm{BV}(Q)$ is disjoint from $\mathrm{FV}(Q)$ as required. ■

Again this makes the following definition much simpler since variable capture does not become a problem. And again we use an operator to encode "cyclic scope nesting."

4.3 Definition. The *$\lambda\mu$-reduction steps* are:

$$(\beta\mu) \qquad ((\lambda x.M\mu)\rho\ S)\gamma \Rightarrow M\,(\mu \circledast \rho \circledast \langle\!\langle x := S \rangle\!\rangle \circledast \gamma)$$

$$(\mu_1) \qquad x\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \Rightarrow x\langle\!\langle y_2 := M_2(\mu_2 \circledast \langle\!\langle y_1 := M_1\mu_1 \rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{if } x \neq y_1 \wedge \langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$$

$$(\mu_2) \qquad y_1\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \Rightarrow M_1\,(\mu_1 \circledast \langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) \quad \text{with } \mathrm{DV}(\mu_1) \cap \mathrm{FV}(M_2\mu_2\ldots M_k\mu_k) = \varnothing$$

$$(\mu_3) \qquad (\lambda x.M\mu)\,\rho \Rightarrow (\lambda x.M(\mu \circledast \rho))\,\langle\!\langle\rangle\!\rangle \quad \text{if } \rho \not\equiv \langle\!\langle\rangle\!\rangle$$

$$(\mu_4) \qquad (M\mu\ P\pi)\,\rho \Rightarrow (M(\mu \circledast \rho)\ P(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle \quad \text{if } \rho \not\equiv \langle\!\langle\rangle\!\rangle$$

where the *cyclic substitution nesting* operator, $\circledast$, is defined as follows for explicit substitutions $\mu = \langle\!\langle x_1 := S_1, \ldots_j \rangle\!\rangle$ and $\rho = \langle\!\langle y_1 := T_1, \ldots_k \rangle\!\rangle$ when $x_1, \ldots \neq y_1, \ldots$ and $x_1, \ldots \notin \mathrm{FV}(T_1 \ldots)$:

$$\mu \circledast \rho \equiv \langle\!\langle x_1 := S_1, \ldots_j , y_1 := T_1, \ldots_k \rangle\!\rangle$$

The theory $\boldsymbol{\mu}$ contains the least stable, compatible relation $\Rightarrow$ satisfying the μ-rules $(\mu_1 \ldots \mu_4)$; $\boldsymbol{\lambda\mu}$ is $(\beta\mu) + \boldsymbol{\mu}$.

Remarks.

(i) (μ_1, μ_2) encode that μ-binding is sticky: (μ_1) removes a binding, but only to have it appear inside the other bindings; the second side condition is to prevent the relation from being reflexive. (μ_2) unfolds and duplicates a binding. This "change of focus" idea is reminiscent of graph reduction; we comment further on that in the conclusion.

(ii) (μ_3, μ_4) are exactly as for the $\lambda\sigma$-rules with side conditions to ensure that $\Rightarrow$ is not reflexive.

(iii) As for $\lambda\sigma$ the variable convention 2.4 ensures that *variable capture* does not occur and that all uses of $\circledast$ are well defined, *except* in (μ_2) where we explicitly require that $M_1\mu_1$ be on a form such that $\circledast$ is defined and there is no capture of the free variables at the level the bindings are moved out to (this is clearly always possible).

(iv) The following variant of (μ_2) could be used because of the scope rules:

$$y_1\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \Rightarrow M_1\,(\mu_1 \circledast \langle\!\langle y_1 := M_1\langle\!\langle\rangle\!\rangle, y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle) \quad \text{with } \mathrm{DV}(\mu_1) \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$$

It utilises the fact that the inner occurrence of μ_1 in (μ_2) defines nothing new in the sense that all the same definitions are present verbatim in the surrounding scope. We refrain from this to be able to handle $M_1\mu_1$ as a unit.

4.4 Example. The **letrec** example of the introduction may be coded as a $\lambda\mu$-term by

$$\mathit{fib}(n) \equiv (f\,0\,1\,n)\langle\!\langle f := \lambda n_0 n_1 n'.(\mathbf{if}\ n' = 0\ \mathbf{then}\ n_0\ \mathbf{else}\ f\,n_1\,(n_0 + n_1)\,(n' - 1))\rangle\!\rangle$$

assuming suitable definitions of arithmetic and **if-then-else**.

We again state the properties of the nesting operator for use in proofs.

4.5 Propositions.

(i) *Cyclic nesting is associative:* $(\mu \circledast \rho) \circledast \gamma \equiv \mu \circledast (\rho \circledast \gamma)$.

(ii) *Cyclic nesting is commutative:* $\mu \circledast \rho \equiv \rho \circledast \mu$.

Proofs. Follow immediately from definition 4.3. ■

Remark. Notice how the proofs are much simpler than their $\lambda\sigma$-counterparts. This is because nesting depends on the definition of FV (definition 4.1(iv)) which is symmetric in body and substituend terms in $\lambda\mu$ but not in $\lambda\sigma$. This is also why $\circledast$ is commutative (as $*$ was not).

$\boldsymbol{\mu}$ *and cyclic substitution*

We now establish the confluence of $\boldsymbol{\mu}$ and show how **letrec** is simulated. Since $\boldsymbol{\mu}$ is obviously not strongly normalising it is not sufficient to prove local confluence but we must provide a "strip lemma" (*cf.* proposition 2.2(iii)) as well. Fortunately this is relatively simple with $\boldsymbol{\mu}$ using the standard "mark to trace residuals" technique.

4.6 Proposition. $\boldsymbol{\mu}$ *is locally confluent.*

Proof. As for proposition 3.6(ii) we show that each critical pair has a common reduct; most of the cases are simplified by using the associativity and commutivity of $\circledast$ freely.

Case (μ_1) containing (μ_1): Assume $x \neq y_1$, $\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$, $z \neq w_1$, and $\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$.

$$x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := \underline{z\,\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle}, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle$$

$$x\,\langle\!\langle y_2 := z\langle\!\langle w_1 := N_1\gamma_1, \ldots_n, y_1 := M_1\mu_1\rangle\!\rangle, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \text{(}\mu_1\text{)} \quad x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := z\,\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle), \ldots_n \rangle\!\rangle, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \text{(}\mu_1\text{)}$$

$$x\,\langle\!\langle y_2 := z\,\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle), \ldots_n, y_1 := M_1\mu_1\rangle\!\rangle, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \checkmark$$

Case (μ_1) containing (μ_2): Assume $x \neq y_1$, $\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$, and $\mathrm{DV}(\gamma_1) \cap \mathrm{FV}(N_2\gamma_2 \ldots N_n\gamma_n) = \varnothing$.

$$x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := \underline{z_1\,\langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle}, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle$$

$$x\,\langle\!\langle y_2 := z_1\,\langle\!\langle z_1 := N_1\gamma_1, \ldots_n, y_1 := M_1\mu_1\rangle\!\rangle, y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{(}\mu_2\text{)} \quad x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle), y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \text{(}\mu_2\text{)}$$

$$x\,\langle\!\langle y_2 := N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n, y_1 := M_1\mu_1\rangle\!\rangle), y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \checkmark$$

Case (μ_1) containing (μ_3): Assume $x \neq y_1$, $\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$, and $\rho \not\equiv \langle\!\langle\rangle\!\rangle$.

$$x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := \underline{(\lambda z.P\pi)\rho}, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle$$

$$x\,\langle\!\langle y_2 := (\lambda z.P\pi)\,(\rho \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{(}\mu_3\text{)} \quad x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := (\lambda z.P(\pi \circledast \rho))\langle\!\langle\rangle\!\rangle, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \text{(}\mu_3\text{)}$$

$$x\,\langle\!\langle y_2 := (\lambda z.P(\pi \circledast \rho \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle))\langle\!\langle\rangle\!\rangle, y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \checkmark$$

Case (μ_1) containing (μ_4): Assume $x \neq y_1$, $\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$, and $\rho \not\equiv \langle\!\langle\rangle\!\rangle$.

$$x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := \underline{(P\pi\,Q\gamma)\rho}, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle$$

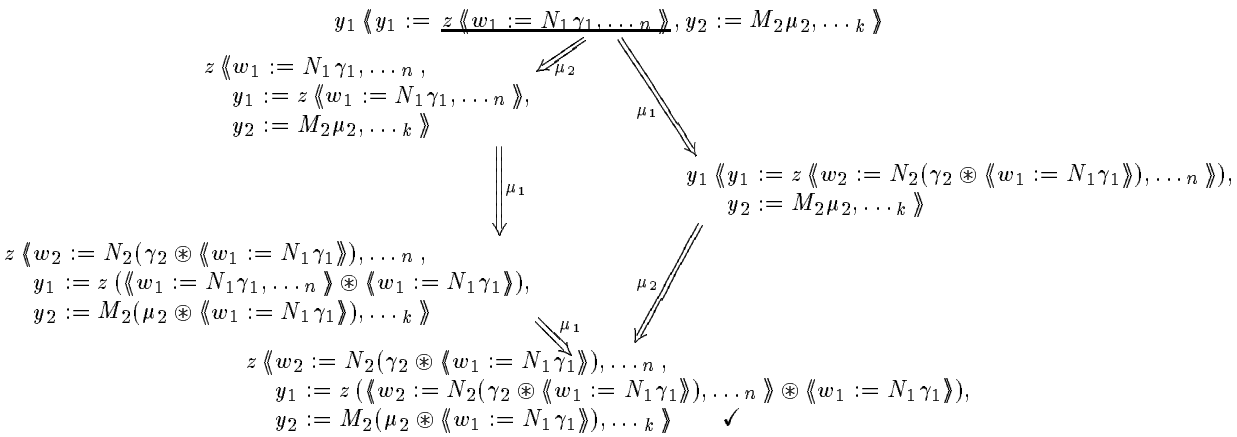
$$x\,\langle\!\langle y_2 := (P\pi\,Q\gamma)\,(\rho \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{(}\mu_1\text{)} \quad x\,\langle\!\langle y_1 := M_1\mu_1, y_2 := (P(\pi \circledast \rho)\,Q(\gamma \circledast \rho))\langle\!\langle\rangle\!\rangle, y_3 := M_3\mu_3, \ldots_k \rangle\!\rangle \quad \text{(}\mu_4\text{)}$$

$$x\,\langle\!\langle y_2 := (P(\pi \circledast \rho)\,Q(\gamma \circledast \rho))\langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle, y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{(}\mu_1\text{)} \quad \text{(}\mu_4\text{)}$$

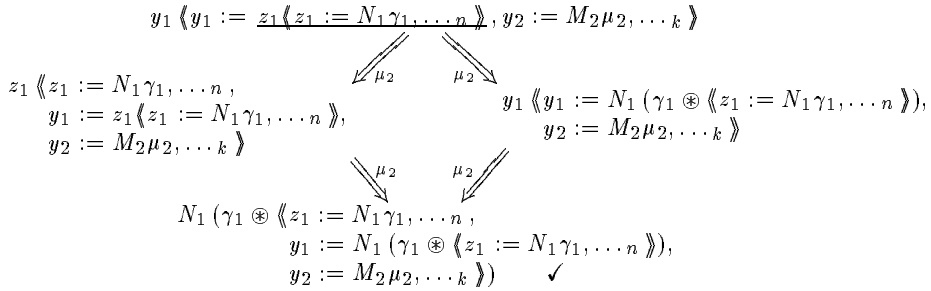
$$x\,\langle\!\langle y_2 := (P(\pi \circledast \rho \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle)\,Q(\gamma \circledast \rho \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle))\langle\!\langle\rangle\!\rangle, y_3 := M_3(\mu_3 \circledast \langle\!\langle y_1 := M_1\mu_1\rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \checkmark$$

Case (μ_2) containing (μ_1): Assume $\{w_1, \ldots, w_n\} \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$, $z \neq w_1$,

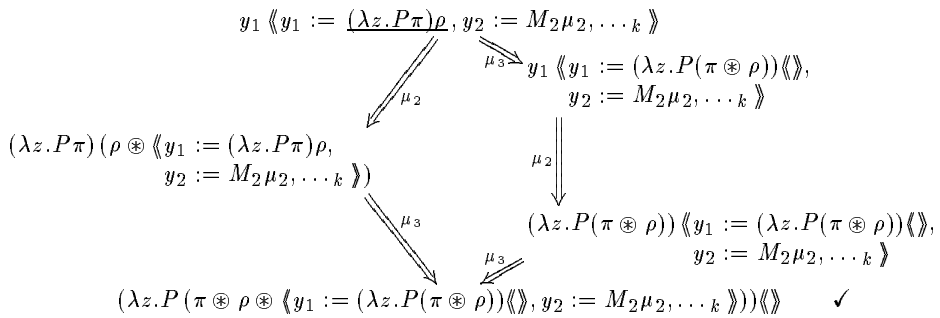
and $\langle\!\langle w_1 := N_1\gamma_1, \ldots n \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$.



Case (μ_2) containing (μ_2): Assume $\{z_1, \ldots, z_n\} \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$ and equivalently $\mathrm{DV}(\gamma_1) \cap \mathrm{FV}(N_2\gamma_2 \ldots N_n\gamma_n) = \varnothing$.



Case (μ_2) containing (μ_3): Assume $\mathrm{DV}(\rho) \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$ and $\rho \not\equiv \langle\!\langle\rangle\!\rangle$.



Case (μ_2) containing (μ_4): Assume $\mathrm{DV}(\rho) \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$ and $\rho \not\equiv \langle\!\langle\rangle\!\rangle$.

$$
\begin{array}{c}
y_1 \langle\!\langle y_1 := \underline{(P\pi\, Q\gamma)\rho}, y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle \\
\swarrow\!\!\swarrow{\scriptstyle \mu_2} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_4} \\
(P\pi\, Q\gamma)\,(\rho \circledast \langle\!\langle y_1 := (P\pi\, Q\gamma)\rho, \quad y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle) \qquad y_1 \langle\!\langle y_1 := (P(\pi \circledast \rho)\ Q(\gamma \circledast \rho))\langle\!\langle\rangle\!\rangle, \quad y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle \\
\Downarrow{\scriptstyle \mu_4} \qquad\qquad {\scriptstyle \mu_2}\Downarrow \\
(P\pi\, Q\gamma)\,(\rho \circledast \langle\!\langle y_1 := (P(\pi \circledast \rho)\ Q(\gamma \circledast \rho))\langle\!\langle\rangle\!\rangle, \quad y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle) \qquad (P(\pi \circledast \rho)\ Q(\gamma \circledast \rho))\,\langle\!\langle y_1 := (P(\pi \circledast \rho)\ Q(\gamma \circledast \rho))\langle\!\langle\rangle\!\rangle, \quad y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle \\
\searrow\!\!\searrow{\scriptstyle \mu_4} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_4} \\
(P(\pi \circledast \rho \circledast \langle\!\langle y_1 := (P\pi\, Q\gamma)\rho, y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle) \\
Q(\pi \circledast \rho \circledast \langle\!\langle y_1 := (P\pi\, Q\gamma)\rho, y_2 := M_2\mu_2, \ldots_k \rangle\!\rangle))\langle\!\langle\rangle\!\rangle \qquad \checkmark
\end{array}
$$

Case (μ_3) containing (μ_1): Assume $\rho \not\equiv \langle\!\langle\rangle\!\rangle$, $z \neq w_1$, and $\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$.

$$
\begin{array}{c}
(\lambda x.\, \underline{z \langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle})\,\rho \\
\swarrow\!\!\swarrow{\scriptstyle \mu_3} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_1} \\
(\lambda x.z(\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho))\,\langle\!\langle\rangle\!\rangle \qquad\qquad (\lambda x.z\,\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle)\ldots_n \rangle\!\rangle)\,\rho \\
\searrow\!\!\searrow{\scriptstyle \mu_1} \qquad\qquad \swarrow\!\!\swarrow{\scriptstyle \mu_3} \\
(\lambda x.z\,\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle)\ldots_n \rangle\!\rangle \circledast \rho)\,\langle\!\langle\rangle\!\rangle \qquad \checkmark
\end{array}
$$

Case (μ_3) containing (μ_2): Assume $\rho \not\equiv \langle\!\langle\rangle\!\rangle$ and $\mathrm{DV}(\gamma_1) \cap \mathrm{FV}(N_2\gamma_2 \ldots N_n\gamma_n) = \varnothing$.

$$
\begin{array}{c}
(\lambda x.\, \underline{z_1 \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle})\,\rho \\
\swarrow\!\!\swarrow{\scriptstyle \mu_3} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_2} \\
(\lambda x.z_1(\langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho))\,\langle\!\langle\rangle\!\rangle \qquad\qquad (\lambda x.N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle))\,\rho \\
\searrow\!\!\searrow{\scriptstyle \mu_2} \qquad\qquad \swarrow\!\!\swarrow{\scriptstyle \mu_3} \\
(\lambda x.N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho))\,\langle\!\langle\rangle\!\rangle \qquad \checkmark
\end{array}
$$

Case (μ_3) containing (μ_3): As in proof of 3.6(ii). ✓

Case (μ_3) containing (μ_4): As in proof of 3.6(ii). ✓

Case (μ_4) containing (μ_1): Assume $\rho \not\equiv \langle\!\langle\rangle\!\rangle$, $z \neq w_1$, and $\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$.

$$
\begin{array}{c}
(\,\underline{z \langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle}\ N\pi)\,\rho \\
\swarrow\!\!\swarrow{\scriptstyle \mu_4} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_1} \\
(z\,(\langle\!\langle w_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho)\ N(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle \qquad\qquad (z\,\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle), \ldots_n \rangle\!\rangle\ N\pi)\,\rho \\
\searrow\!\!\searrow{\scriptstyle \mu_1} \qquad\qquad \swarrow\!\!\swarrow{\scriptstyle \mu_4} \\
(z\,(\langle\!\langle w_2 := N_2(\gamma_2 \circledast \langle\!\langle w_1 := N_1\gamma_1\rangle\!\rangle), \ldots_n \rangle\!\rangle \circledast \rho)\ N(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle
\end{array}
$$

and symmetrically for the right subterm. ✓

Case (μ_4) containing (μ_2): Assume $\rho \not\equiv \langle\!\langle\rangle\!\rangle$.

$$
\begin{array}{c}
(\,\underline{z_1 \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle}\ P\pi)\,\rho \\
\swarrow\!\!\swarrow{\scriptstyle \mu_4} \qquad\qquad \searrow\!\!\searrow{\scriptstyle \mu_2} \\
(z_1(\langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho)\ P(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle \qquad\qquad (N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle)\ P\pi)\,\rho \\
\searrow\!\!\searrow{\scriptstyle \mu_2} \qquad\qquad \swarrow\!\!\swarrow{\scriptstyle \mu_4} \\
(N_1\,(\gamma_1 \circledast \langle\!\langle z_1 := N_1\gamma_1, \ldots_n \rangle\!\rangle \circledast \rho)\ P(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle
\end{array}
$$

and symmetrically for the right subterm. ✓

Case (μ_4) containing (μ_3): As in proof of 3.6(ii). ✓

Case (μ_4) containing (μ_4): As in proof of 3.6(ii). ✓

where "as in proof of 3.6(ii)" of course implies that μs should be replaced by σs. ■

4.7 Definition.

(i) The $\overline{\lambda\mu}$*-terms*, are given as in definition 4.1 with the change that we introduce special *overlined redexes* on the form $\overline{M}\langle\!\langle \ldots \rangle\!\rangle$ with the associated reduction rules

$$(\overline{\mu}_1) \quad \overline{x}\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \Rightarrow x\,\langle\!\langle y_2 := M_2(\mu_2 \circledast \langle\!\langle y_1 := M_1\mu_1 \rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{if } x \neq y_1$$

$$(\overline{\mu}_2) \quad \overline{y}_1\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \Rightarrow M_1\,(\mu_1 \circledast \langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) \quad \text{with } \mathrm{DV}(\mu_1) \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$$

$$(\overline{\mu}_3) \quad \overline{(\lambda x.M\mu)}\,\rho \Rightarrow (\lambda x.M(\mu \circledast \rho))\,\langle\!\langle\rangle\!\rangle \quad \text{if } \rho \not\equiv \langle\!\langle\rangle\!\rangle$$

$$(\overline{\mu}_4) \quad \overline{(M\mu\ P\pi)}\,\rho \Rightarrow (M(\mu \circledast \rho)\ P(\pi \circledast \rho))\,\langle\!\langle\rangle\!\rangle \quad \text{if } \rho \not\equiv \langle\!\langle\rangle\!\rangle$$

that thus remove one overline. The set of opaque overlined terms is $\overline{\Lambda\mu}$; the theory $\boldsymbol{\overline{\mu}}$ is just $(\overline{\mu}_1 \ldots \overline{\mu}_4)$ and the theory $\boldsymbol{\mu\overline{\mu}}$ furthermore allows $(\mu_1 \ldots \mu_4)$ (just copying overlines).

(ii) If $S \in \overline{\Lambda\mu}$ then $|S| \in \Lambda\mu$ is obtained by removing all overlines from S.

(iii) If $S \in \overline{\Lambda\mu}$ then $\varphi(S) \in \Lambda\mu$ is obtained by rewriting all $\overline{\boldsymbol{\mu}}$-redexes of S inside out. Inductively this means

$$\varphi(\,x\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = x\,\langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle$$

$$\varphi(\,(\lambda x.P\pi)\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = (\lambda x.P'\pi')\,\langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle$$

$$\varphi(\,(P\pi\,Q\rho)\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = (P'\pi'\,Q'\rho')\,\langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle$$

$$\varphi(\,\overline{x}\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = x\,\langle\!\langle y_2 := M_2'(\mu_2' \circledast \langle\!\langle y_1 := M_1'\mu_1' \rangle\!\rangle), \ldots_k \rangle\!\rangle \quad \text{if } x \neq y_1$$

$$\varphi(\,\overline{y}_1\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = M_1'\,(\mu_1' \circledast \langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle) \quad \text{with } \mathrm{DV}(\mu_1') \cap \mathrm{FV}(M_2'\mu_2' \ldots M_k'\mu_k') = \varnothing$$

$$\varphi(\,\overline{(\lambda x.P\pi)}\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = (\lambda x.M'(\mu' \circledast \langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle))\,\langle\!\langle\rangle\!\rangle \quad \text{if } k \geq 1$$

$$\varphi(\,\overline{(M\mu\ P\pi)}\,\langle\!\langle y_1 := M_1\mu_1, \ldots_k \rangle\!\rangle) = (M'(\mu' \circledast \langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle)\ P'(\pi' \circledast \langle\!\langle y_1 := M_1'\mu_1', \ldots_k \rangle\!\rangle))\,\langle\!\langle\rangle\!\rangle \quad \text{if } k \geq 1$$

where for all pairs $M\mu$ on each LHS we assume a side condition "where $M'\mu' \equiv \varphi(M\mu)$" on the corresponding RHS.

Remark. Clearly $\Lambda\mu \subseteq \overline{\Lambda\mu}$.

The use of overlines gives the following crucial properties which we will use for the "tracing" in the strip lemma.

4.8 Propositions.

(i) $\boldsymbol{\mu\overline{\mu}}$ *is locally confluent.*
(ii) $\overline{\boldsymbol{\mu}} \vdash S \twoheadrightarrow \varphi(S)$ *for all* $S \in \overline{\Lambda\mu}$.
(iii) *For* $S, T \in \overline{\Lambda\mu}$ *we may complete*

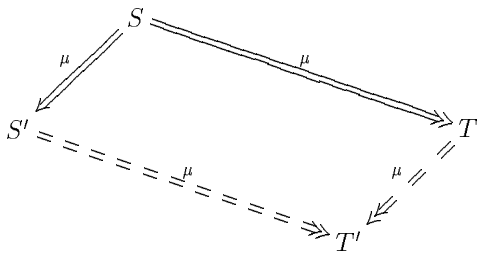
$$\begin{array}{ccc} S & \xRightarrow[\mu\overline{\mu}]{} & T \\ \| & & \| \\ \|_{\overline{\mu}} & & \|_{\overline{\mu}} \\ \Downarrow & & \Downarrow \\ S' & =\!=\!\Rightarrow_{\overline{\mu}} & T' \end{array} \qquad \text{with } S', T' \in \Lambda\mu$$

Proofs.

(i) Follows from the local confluence of $\boldsymbol{\mu}$ and the fact that the $\overline{\mu}$-rules duplicate all $\boldsymbol{\mu}$-rules and no rule ever 'splits' an overline. ✓
(ii) By induction over the structure of S: if $S \twoheadrightarrow S'$ reduces all subterms of S then either $S' \equiv \varphi(S)$ or $S' \Rightarrow \varphi(S)$. ✓
(iii) From (i) and (ii). ■

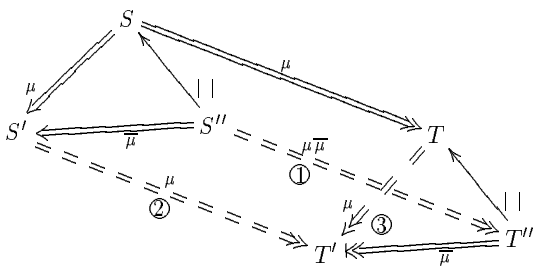
With this we are able to prove a strip lemma for $\boldsymbol{\mu}$.

4.9 Proposition ("strip lemma").



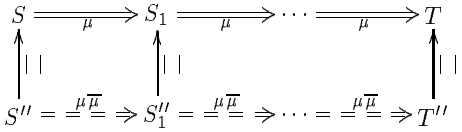
Proof. We proceed as for the λ-calculus [4, §11.1] (except we use "overlining" to keep track of residuals rather than indexes).

First let $S'' \in \overline{\Lambda\mu}$ be S but with the substitution of the redex reduced by the reduction $\boldsymbol{\mu} \vdash S \Rightarrow S'$ overlined such that $\overline{\boldsymbol{\mu}} \vdash S'' \Rightarrow S'$. Then the following diagram proves the strip lemma:



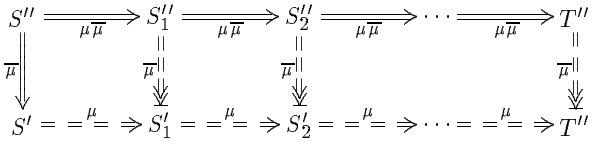
where we will explain how each of the numbered reductions were constructed:

① This follows from the completion of



where each cell is constructed by simply reducing (rightward) the same redex of the overlined term as of the primed one, possibly by one of the added rules (when reducing a residual of the original overlined redex)—none of these reductions will affect the possibility of using $|\ |$ to get rid of the remaining residuals, though. ✓

② This follows from ① and the induction



where the step is proposition 4.8(iii)—the leftmost is a single reduction only because we know that S'' only has one overlined redex. ✓

③ $\overline{\boldsymbol{\mu}}$ reduces only marked redices. This may also be done by first unmarking them with $|\ |$ and then reducing the same redices with the corresponding $\boldsymbol{\mu}$-rules. ■

4.10 Proposition. $\boldsymbol{\mu}$ *is confluent.*

Proof. Follows by induction 2.2(iii) over the strip lemma 4.9. ■

Finally we show that $\boldsymbol{\mu}$ implements mutual recursive unfolding as described at the beginning of this section. We can not define sinking as for $\lambda\sigma$ since "unfolding all cyclic bindings" is most likely to be undefined (*i.e.*, yield an infinite term). But we can show that $\boldsymbol{\mu}$ simulates **letrec** by showing that it satisfies the unfolding property given in the start of this section.

4.11 Definition. *Lifting* of λ-terms to $\lambda\mu$-terms, $\ulcorner\_\urcorner\colon \Lambda \to \Lambda\mu$, is defined to add dummy substitutions everywhere:

$$\begin{aligned} \ulcorner x \urcorner &= x\,\langle\rangle \\ \ulcorner \lambda x.M \urcorner &= (\lambda x.\ulcorner M\urcorner)\,\langle\rangle \\ \ulcorner M\,N \urcorner &= (\ulcorner M\urcorner\ \ulcorner N\urcorner)\,\langle\rangle \end{aligned}$$

We will write $\ulcorner M\urcorner\mu$ for $M'\mu$ where $M'\langle\rangle = \ulcorner M\urcorner$.

4.12 Lemma. $\boldsymbol{\mu}$ *simulates* **letrec***-unfolding:*

$$\begin{aligned} &\ulcorner M\urcorner\langle\!\langle x_1 := \ulcorner N_1\urcorner, \ldots_k \rangle\!\rangle \\ &\qquad \twoheadrightarrow M[x_1 := \ulcorner N_1\urcorner\langle\!\langle x_1 := \ulcorner N_1\urcorner, \ldots_k \rangle\!\rangle, \ldots, x_k := \ulcorner N_k\urcorner\langle\!\langle x_1 := \ulcorner N_1\urcorner, \ldots_k \rangle\!\rangle] \end{aligned}$$

Proof. The trick is to prove that it is possible to 'run' each binder in the list once. But this is trivial: we just have to distribute the substitution down into every component of the term as done by the following function, passing down and overlining each variable to be replaced:

$$\begin{aligned}
\psi(\,x\,\mu) &= \overline{x}\mu && \text{if } x \in \{x_1,\ldots,x_k\}\\
\psi(\,x\,\mu) &= x\mu && \text{if } x \notin \{x_1,\ldots,x_k\}\\
\psi(\,(\lambda x.M\langle\!\rangle)\,\mu) &= (\lambda x.\psi(M\mu))\,\langle\!\rangle\\
\psi(\,(M\langle\!\rangle\,N\langle\!\rangle)\,\mu) &= (\psi(M\mu)\,\psi(N\mu))\,\langle\!\rangle
\end{aligned}$$

(where we make use of the variable convention to pass λs and the fact that $x_1,\ldots,x_k$ are the only μ-bound variables). Then we may use φ from definition 4.7(iii) to substitute the variables exactly once:

$$\varphi(\psi(\ulcorner M\urcorner\langle\!\langle x_1 := \ulcorner N_1\urcorner,\ldots_k\rangle\!\rangle)) \quad\equiv\quad M[x_1 := \ulcorner N_1\urcorner\langle\!\langle x_1 := \ulcorner N_1\urcorner,\ldots_k\rangle\!\rangle,\ldots_k] \quad\blacksquare$$

Remark. An alternate way of showing this is to consider the infinite unfolding of mutual recursion equivalent to infinite regular terms as discussed by Courcelle [7]. This is outside the scope of this paper.

4.13 Lemma. $\boldsymbol{\mu} \models \boldsymbol{\sigma}$.

Proof. Follows from confluency of $\boldsymbol{\mu}$ using the straight forward representation

$$\begin{aligned}
\chi(x) &= x\\
\chi(\lambda x.S) &= \lambda x.\chi(S)\\
\chi(ST) &= \chi(S)\,\chi(T)\\
\chi(M\,\langle x_1 := S_1,\ldots_k\rangle) &= \chi(M)\,\langle\!\langle x_1 := \chi(S_1),\ldots_k\rangle\!\rangle
\end{aligned}$$

requiring $x_1,\ldots,x_k \notin \mathrm{FV}(S_1\ldots S_k)$, since then clearly

$$\begin{aligned}
&\chi(M\langle x_1 := S_1,\ldots_k\rangle)\\
&\qquad \twoheadrightarrow M[x_1 := \chi(S_1)\langle\!\langle x_1 := \chi(S_1),\ldots_k\rangle\!\rangle,\ldots,x_k := \chi(S_k)\langle\!\langle x_1 := \chi(S_1),\ldots_k\rangle\!\rangle]\\
&\qquad \equiv M[x_1 := \chi(S_1),\ldots,x_k := \chi(S_k)]
\end{aligned}$$

because there are no free $\{x_1,\ldots,x_k\}$ in the S_i. $\blacksquare$

$\boldsymbol{\lambda\mu}$ *may simulate* $\boldsymbol{\lambda\sigma}$

We first establish that $\boldsymbol{\lambda\mu}$ is confluent. It is not difficult to use this to prove that $\boldsymbol{\lambda\mu}$ is a λ-model using the same approach as for $\boldsymbol{\lambda\sigma}$. This is not very interesting, however, since nothing new is learned. We will instead prove that $\boldsymbol{\lambda\mu}$ is a "$\lambda\sigma$-model," *i.e.*, that cyclic substitution may simulate acyclic substitution.

4.14 Lemma. $\boldsymbol{\lambda\mu}$ *is confluent.*

Proof. Very similar to the proof of lemma 3.9: $(\beta\mu)$ is obviously confluent and $\boldsymbol{\mu}$ is confluent by proposition 4.10 so we use Hindley-Rosen lemma 2.2(ii) and just show that each single $\boldsymbol{\mu}$-rule commutes with it.

Case Shared $(\beta\mu)$ and (μ_4) redex: As in the proof of 3.9.

Case $(\beta\mu)$ containing (μ_1): Assume $z \neq w_1$, and $\langle\!\langle w_1 := M_1\mu_1, \ldots_n \rangle\!\rangle \not\equiv \langle\!\langle\rangle\!\rangle$.

$$((\lambda x.\, \underline{z\,\langle\!\langle w_1 := M_1\mu_1, \ldots_n \rangle\!\rangle})\rho\, S)\,\gamma$$

$$\swarrow_{\beta\mu} \qquad \searrow_{\mu_1}$$

$$z\,(\langle\!\langle w_1 := M_1\mu_1, \ldots_n \rangle\!\rangle \circledast \rho \circledast \langle\!\langle x := S\rangle\!\rangle \circledast \gamma) \qquad ((\lambda x.z\,\langle\!\langle w_2 := M_2(\mu_2 \circledast \langle\!\langle w_1 := M_1\mu_1\rangle\!\rangle), \ldots_n \rangle\!\rangle)\rho\, S)\,\gamma$$

$$\searrow_{\mu_1} \qquad \swarrow_{\beta\mu}$$

$$z\,(\langle\!\langle w_2 := M_2(\mu_2 \circledast \langle\!\langle w_1 := M_1\mu_1\rangle\!\rangle), \ldots_n \rangle\!\rangle \circledast \rho \circledast \langle\!\langle x := S\rangle\!\rangle \circledast \gamma)$$

where the lower right $(\beta\mu)$ reduction makes use of the garbage collection equivalence on each of the binders in ρ, $\langle\!\langle x := S\rangle\!\rangle$, and γ, since we know from the variable convention that w_1 does not occur free in any of them and that $M_1\mu_1$ does not contain any of their defined variables as free. ✓

Case $(\beta\mu)$ containing (μ_2): Assume $\mathrm{DV}(\mu_1) \cap \mathrm{FV}(M_2\mu_2 \ldots M_k\mu_k) = \varnothing$.

$$((\lambda x.\, \underline{z_1\langle\!\langle z_1 := M_1\mu_1, \ldots_k \rangle\!\rangle})\rho\, S)\,\gamma$$

$$\swarrow_{\beta\mu} \qquad \searrow_{\mu_2}$$

$$z_1\,(\langle\!\langle z_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \circledast \rho \circledast \langle\!\langle x := S\rangle\!\rangle \circledast \gamma\rangle\!\rangle) \qquad ((\lambda x.M_1\,(\mu_1 \circledast \langle\!\langle z_1 := M_1\mu_1, \ldots_k \rangle\!\rangle))\rho\, S)\,\gamma$$

$$\searrow_{\mu_2} \qquad \swarrow_{\beta\mu}$$

$$M_1\,(\mu_1 \circledast \langle\!\langle z_1 := M_1\mu_1, \ldots_k \rangle\!\rangle \circledast \rho \circledast \langle\!\langle x := S\rangle\!\rangle \circledast \gamma\rangle\!\rangle) \qquad \checkmark$$

Case $(\beta\mu)$ containing (μ_3): As in the proof of 3.9. ✓

Case $(\beta\mu)$ containing (μ_4): As in the proof of 3.9. ✓

Case (μ_3) containing $(\beta\mu)$: As in the proof of 3.9. ✓

Case (μ_4) containing $(\beta\mu)$: As in the proof of 3.9. ✓

where "as in proof of 3.9" of course implies that μs should be replaced by σs. ■

4.15 Theorem. $\boldsymbol{\lambda\mu} \models \boldsymbol{\lambda\sigma}$.

Proof. This is trivial by extending lemma 4.13 to include $(\beta\sigma)$ and $(\beta\mu)$ since they are syntactically the same (though one is with $\langle\rangle$s and the other with $\langle\!\langle\rangle\!\rangle$s, of course). ■

4.16 Corollary. $\boldsymbol{\lambda\mu} \models \boldsymbol{\lambda}$.

Proof. By theorem 3.10 and 4.15. ■

Summary

We have shown how $\boldsymbol{\lambda\mu}$ provides an explicit operational model for the λ-calculus enriched with **letrec** as used in functional programming languages using the identification

$$\ulcorner \mathbf{letrec}\ x_1 = N_1\ \mathbf{and} \ldots \mathbf{and}\ x_k = N_k\ \mathbf{in}\ M \urcorner \quad \equiv \quad \ulcorner M \urcorner \langle\!\langle x_1 := \ulcorner N_1 \urcorner, \ldots_k \rangle\!\rangle$$

because by lemma 4.12 and corollary 4.16 we have

$$\begin{aligned}
\boldsymbol{\lambda\sigma} \quad \vdash \quad & \ulcorner (\lambda x.M)N \urcorner \quad \twoheadrightarrow \quad \ulcorner M[x := N] \urcorner \\
& \ulcorner \mathbf{letrec}\ x_1 = N_1\ \mathbf{and} \ldots \mathbf{and}\ x_k = N_k\ \mathbf{in}\ M \urcorner \\
& \quad \twoheadrightarrow \quad M[\ x_1 := \ulcorner \mathbf{letrec}\ x_1 = N_1\ \mathbf{and} \ldots \mathbf{and}\ x_k = N_k\ \mathbf{in}\ N_1 \urcorner, \\
& \qquad\qquad \vdots \\
& \qquad\qquad x_k := \ulcorner \mathbf{letrec}\ x_1 = N_1\ \mathbf{and} \ldots \mathbf{and}\ x_k = N_k\ \mathbf{in}\ N_k \urcorner\ ]
\end{aligned}$$

5 Conclusion and Future Work

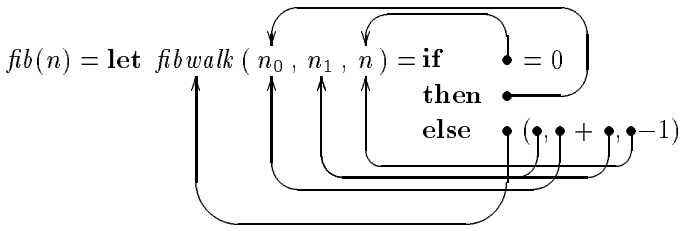
In this section we conclude by summarising the main results and pointing out applications and directions for future work, in particular with respect to describing graph reduction.

Summary

We have discussed how to describe sharing and recursion in conditional term rewrite models of λ-calculi, and have shown proofs of confluency. The knowledge gained is that *explicitly represented sharing and recursion* can be maintained and exploited in a computational model. We hope with this to contribute to the creation of formal links between the 'operational insight' of implementors and the 'denotational oversight' of formalists.

What next?

The presentation above has made it clear how the concepts of *sharing*, *cycles*, and *garbage collection* as found in functional graph reduction systems are related to *variable* and *substitution* as found in λ-calculus, and in particular where the latter needs to be extended. One particularly interesting direction is what can be modeled by combining explicit substitution with various forms of *labeling* of terms in the style of Lévy [13, 14]. It seems to be possible to use this to refine the definition of *unfolding* used to be that of *graph traversal*: the example in the introduction should intuitively be represented directly as



using labels to indicate the uniqueness of the arrows, enriching the substitution rules to effectively do *graph substitution*, *i.e.*, term graph rewriting [19]. This way the 'focus' can change without changing the term. This is the subject of current work.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy, *Explicit Substitutions*, in "POPL '90—Seventeenth Annual ACM Symposium on Principles of Programming Languages" (San Francisco, California), January 1990, pp. 31–46.

2. S. Abramsky, *The Lazy Lambda Calculus*, ch. 4, in D. A. Turner (ed.), "Research Topics in Functional Programming," Addison-Wesley, 1990, pp. 65–116.
3. Z. M. Ariola and Arvind, *A Syntactic Approach to Program Transformations*, in "PEPM '91—Symposium on Partial Evaluation and Semantics-based Program Manipulation" (Yale University, New Haven, Connecticut, USA), 17–19 June 1991, pp. 116–129.
4. H. Barendregt, "The Lambda Calculus: Its Syntax and Semantics," revised edition, North-Holland, 1984.
5. H. P. Barendregt, M. C. D. J. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep, *Term Graph Rewriting*, in J. W. de Bakker, A. J. Nijman, and P. C. Treleaven (eds.), "PARLE '87—Parallel Architectures and Languages Europe vol. II" (Eindhoven, The Netherlands), LNCS no. 256, Springer-Verlag, June 1987, pp. 141–158.
6. N. G. de Bruijn, *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation with application to the Church-Rosser theorem*, Koninkijke Nederlandse Akademie van Wetenschappen, Series A, Mathematical Sciences **75** (1972), 381–392.
7. B. Courcelle, *Fundamental Properties of Infinite Trees*, Theoretical Computer Science **25** (1983), 95–169.
8. K. Grue, "Call-by-Mix: A Reduction Strategy for Pure λ-calculus," Unpublished note from DIKU (University of Copenhagen), 1987.
9. C. A. R. Hoare, *Recursive Data Structures*, Journal of Computer and Information Sciences **4** (1975), no. 2, 105–132.
10. G. Huet, *Confluent Reductions: Abstract properties and Applications to Term Rewriting Systems*, Journal of the ACM **27** (1980), no. 4, 797–821.
11. T. Johnsson, *Efficient Compilation of Lazy Evaluation*, SIGPLAN Notices **19** (June 1984), no. 6, 58–69.
12. D. Knuth and P. Bendix, *Simple Word Problems in Universal Algebras*, in J. Leech (ed.), "Computational Problems in Abstract Algebra," Pergamon Press, Elmsford, N.Y., 1970, pp. 263–297.
13. J.-J. Lévy, *Optimal Reductions in the Lambda Calculus*, in J. P. Seldin and J. R. Hindley (eds.), "To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism," Academic Press, 1980, pp. 159–191.
14. L. Maranget, *Optimal Derivations in Weak Lambda-Calculi and in Orthogonal Terms Rewriting Systems*, in "POPL '91—Eightteenth Annual ACM Symposium on Principles of Programming Languages" (Orlando, Florida), January 1991, pp. 255–269.
15. J. McCarthy, *Recursive Functions of Symbolic Expressions*, Communications of the ACM **3** (April 1960), no. 4, 184–195.
16. D. Park, *Finiteness is Mu-ineffable*, Theoretical Computer Science **3** (1976), 173–181.
17. S. L. Peyton Jones, "The Implementation of Functional Programming Languages," Prentice-Hall, 1987.
18. K. H. Rose, *Explicit Cyclic Substitutions*, in M. Rusinowitch and J.-L. Rémy (eds.), "CTRS '92—3rd International Workshop on Conditional Term Rewriting Systems" (Pont-a-Mousson, France), LNCS no. 656, Springer-Verlag, July 1992, pp. 36–50, also available as DIKU semantics note D–143.
19. M. R. Sleep, M. J. Plasmeijer, and M. C. D. J. van Eekelen (eds.), "Term Graph Rewriting: Theory and Practice," John Wiley & Sons, 1993.

20. H. Søndergaard and P. Sestoft, *Referential Transparency, Definiteness and Unfoldability*, Acta Informatica **27** (1990), 505–517.
21. J. Staples, *A Graph-like Lambda Calculus for which Leftmost Outermost Reduction is Optimal*, in V. Claus, H. Ehrig, and G. Rozenberg (eds.), "1978 International Workshop in Graph Grammars and their Application to Computer Science and Biology" (Bad Honnef, F. R. Germany), LNCS no. 73, Springer-Verlag, 1978, pp. 440–454.
22. C. P. Wadsworth, "Semantics and Pragmatics of the Lambda Calculus," Ph.D. Thesis, Programming Research Group, Oxford University, 1971.

Alphabetic concept and symbol index