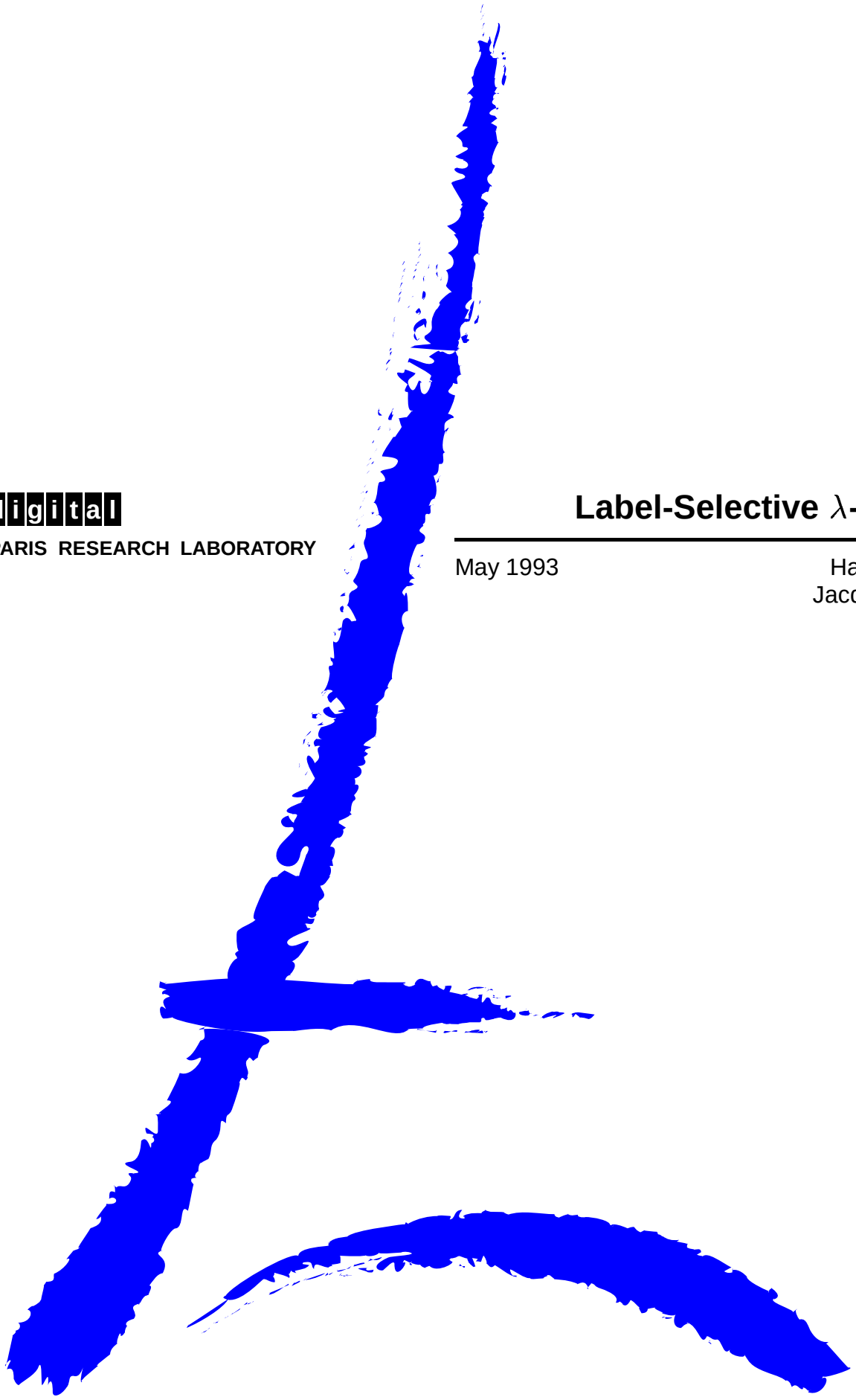


31

digital
PARIS RESEARCH LABORATORY

Label-Selective λ-Calculus

May 1993

Hassan Aït-Kaci
Jacques Garrigue

31

Label-Selective λ-Calculus

Hassan Aït-Kaci
Jacques Garrigue

May 1993

Publication Notes

The authors may be contacted at the following addresses:

Hassan Aït-Kaci

Digital Equipment Corporation
Paris Research Laboratory
85 Avenue Victor Hugo
92500 Rueil-Malmaison, France

hak@prl.dec.com

Jacques Garrigue

Department of Information Science
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo 113, Japan

garrigue@is.s.u-tokyo.ac.jp

Abstract

We introduce an extension of λ-calculus, called *label-selective λ-calculus*, in which arguments of functions are selected by labels. The set of labels includes numeric positions as well as symbolic keywords. While the latter enjoy free commutation, the former must comply with relative precedence in order to preserve currying. This extension of λ-calculus is conservative in the sense that when the set of labels is the singleton $\{1\}$, it coincides with λ-calculus. The main result of this paper is the proof that the label-selective λ-calculus is confluent. In other words, argument selection and reduction commute.

Résumé

Nous présentons une extension du λ-calcul, appelée *λ-calcul label-sélectif*, dans laquelle les arguments des fonctions sont sélectionnés par des étiquettes. L'ensemble des étiquettes comprend des positions numériques aussi bien que des mot-clefs symboliques. Alors que ces derniers jouissent d'une commutativité libre, les premiers obéissent à une précédence relative pour préserver la curryfication. Cette extension du λ-calcul est conservatrice en ce sens que, quand l'ensemble des étiquettes est réduit au singleton $\{1\}$, elle coïncide avec le λ-calcul. Le résultat essentiel de ce papier est la preuve de confluence du λ-calcul label-sélectif. En d'autres termes, la sélection d'arguments et la réduction commutent.

Keywords

λ-Calculus, record calculus, concurrency, communication.

Acknowledgements

The authors wish to thank Kathleen Milsted, Atsushi Ohori, Mitsuhiro Okada, and Andreas Podelski for their comments and suggestions. Also, we are grateful to Patrick Baudelaire and Jean-Christophe Patat for kindly and thoroughly proofreading our manuscript.

Contents

> We can move from one language to another, but in doing so we accept new constraints and make new mistakes. We also adopt a different tone, enjoying the *je ne sais quoi* of *Sprachgefühl.*
>
> ROBERT DARNTON, *The Great Cat Massacre*

1 Synopsis

Many modern programming languages allow specifying arguments of functions and procedures by symbolic keywords as well as using the traditional and natural numeric positions [14, 10, 3]. Symbolic keywords are usually handled as syntactic sugar and "compiled away" as numeric positions. This is made easy if the language does not support currying (like Common LISP or ADA). Even if currying is supported and the situation reduced to numeric positions, it is allowed strictly in a left-to-right order so that the first argument is "consumed" before the second. In general, if a function f is defined on two arguments and it is desired that the second be consumed before the first, one must resort to using an explicit closure of form $\lambda x.\lambda y.f\big(y,x\big)$ and curry that one. However, the cost incurred (the closure construction and ensuing weight of handling in terms of depth of stack, *etc.*) is undue since out-of-order currying simply amounts to commutation of stack offsets.

More precisely, currying is possible thanks to the following natural isomorphism:

$$A \times B \to C \;\simeq\; A \to \big(B \to C\big)$$

for any set A, B and C. However, there is another obvious natural isomorphism that could also be useful; namely, $A \times B \;\simeq\; B \times A$. Hence we should be able to exploit this directly in the form:

$$A \to \big(B \to C\big) \;\simeq\; B \to \big(A \to C\big).$$

One way to do that is to use a style of Cartesian product more of a *category-theoretic*, as opposed to *set-theoretic*, flavor. By this we mean that if projections π_1 and π_2 were used explicitly instead of the implicit 1*st* and 2*nd* of the $\times$ notation, then instead of $A \times B$ we would write $\pi_1 \Rightarrow A \times \pi_2 \Rightarrow B$. Thus, allowing this explicit product expression makes Cartesian product commutative explicitly, as opposed to "up to isomorphism." Indeed, it becomes obvious that:[1]

$$\pi_1 \Rightarrow A \times \pi_2 \Rightarrow B \;\simeq\; \pi_2 \Rightarrow B \times \pi_1 \Rightarrow A,$$

and thus that:

$$\pi_1 \Rightarrow A \to \big(\pi_2 \Rightarrow B \to C\big) \;\simeq\; \pi_2 \Rightarrow B \to \big(\pi_1 \Rightarrow A \to C\big).$$

The advantage of explicit projections is clear: one can account directly for symbolic keywords since these play precisely the role of projections. The other benefit is the

[1] Parse the following with '⇒' binding tighter than '×' or '→'.

aforementioned permutativity of currying which allows out-of-order partial application of function to its arguments. For example, an out-of-order application like $f(2 \Rightarrow a)$ can be readily used when there is a need to consume the second argument before the first, as opposed to the more complex and costly $(\lambda x.\lambda y.f(y,x))(a)$.

The drawback of explicit projections, however, is also obvious: implicit argument positions as numeric offset is lost, and the notation is more cumbersome. It is indeed much easier to write $f(x,y)$ instead of $f(1 \Rightarrow x, 2 \Rightarrow y)$ every time we need to apply f to two arguments.

So the question is: can we allow freely mixing implicit and explicit argument selectors safely? In other words, can we allow the notation $f(x,y)$ to be syntactic sugar for explicitly selecting $f(1 \Rightarrow x, 2 \Rightarrow y)$? If we do, the least we should require is that the "all-functions-are-unary" paradigm of λ-calculus be retained. This means that the equation $f(x,y) = f(x)(y)$ should hold for any such expression. However, the syntactic sugaring gives, on one hand, $f(x,y) = f(1 \Rightarrow x, 2 \Rightarrow y)$, and on the other hand, $f(x)(y) = f(1 \Rightarrow x)(1 \Rightarrow y)$. Therefore the free syntax should guarantee that $f(1 \Rightarrow x, 2 \Rightarrow y) = f(1 \Rightarrow x)(1 \Rightarrow y)$. In other words, stack offset permutation must be built into the rule of application at numeric positions. This is essentially what is performed in the extension of λ-calculus that we propose here.

1.1 Relation to other work

There is an immediate relation between our calculus and the notation with offsets introduced by de Bruijn [5] and used for the compilation of λ-calculus in the style of the SECD machine [9]. In fact, our calculus enforces commutativity of these indices and therefore extends the use of de Bruijn offsets for that model of implementation to include label-selective argument passing. In that way, selective currying can be statically compiled into direct stack access by generating simple arithmetic code involving de Bruijn offsets and selector numbers. Hence, our work is a simple and natural generalization of de Bruijn's idea. We have already adapted the calculus of explicit substitutions [1], and are currently working on a compiling scheme for label-selective λ-calculus based on it.

Another, albeit remote since unexplored, potential connection may be with the recent work of Ohori in compiling extensible records for functional programming [13]. Indeed, records are essentially labeled Cartesian products. Since that style of records allows extensions and out-of-order labels, it is possible to use them in a way similar to ours for passing arguments. At this time, the potential connection is a simple speculation and begs for deeper study.

An intuitive, but accurate, explanation of label-selective λ-calculus can be given as extracting implicit concurrency from λ-calculus. It is well-known that λ-calculus is a sequential calculus and for a clear reason: function application is not commutative. This inherent sequentiality is exacerbated all the more by the strict syntactic left-associativity of application adopted by λ-calculus. Hence, our idea is to reveal the inherent concurrency lost in λ-calculus; namely, commutation of arguments in applications. The syntax and operational semantics that we propose are precisely meant to expose, explicate, and exploit this implicit concurrency. This concurrency is inherent in λ-calculus in the sense that it does not interfere with the confluence of the calculus. This would not be the case with a fully concurrent extension of λ-calculus using parallel composition, a commutative monoid. Thus does our

calculus differ from the known calculi for communication of concurrent processes [4, 12, 11].

In [4], Gérard Boudol proposes γ-calculus, an extension of λ-calculus based on realizing that β-reduction is *communication* between a receiving λ-abstraction and a sending operand along one single channel called λ. Thus, the argument of a β-redex is implicitly prefixed with $\overline{\lambda}$. This idea is taken to its full extent by Robin Milner in [12] where, rather than λ alone, there are (countably) many channel *names*. In both Milner's and Boudol's calculi, parallel composition is used to achieve full concurrency and thus, naturally, confluence is lost. By contrast, label-selective λ-calculus is *not* a fully concurrent calculus. Indeed, our calculus is a confluent one. It explicates the fine interaction between functional application as process communication along channel names that are identified, not as λ's as in [12, 4], but as explicit *position* names. This is a wholly different insight. In addition, the availability of *numeric* channels and their laws of relative commutation allows also to speak of *relatively* numbered channels, as opposed to *absolutely* named channels only.

We are also developing, and will report later [2], our label-selective calculus as a true calculus of communication and concurrency. We plan to extend the calculus along the lines of Robin Milner's π-calculus, adding, for example, process operators, such as parallel composition and non-deterministic choice, as well as exploring other directions, for example, by allowing computable channel names. One of the gains expected is that λ-calculus will need not be *encoded* as in [11], but directly embedded as syntactic identity.

In summary, what we recount in this paper, has not, to our knowledge, been studied as such.

1.2 Organization of paper

We have organized this paper as follows. In Section 2.1 we introduce our language of selective λ-terms. In Section 2.2 we present reduction systems for these terms. The core of the paper lies in Section 3 where we give the proof of confluence of selective λ-calculus. Section 4 is a reflection on the link between symbolic and numeric labels. Finally, we close the paper with some conclusion and a brief discussion of further work to follow this idea in Section 6.

2 Selective λ-terms

2.1 Syntax

Selective λ-terms are formed by variables taken from a set $\mathcal{V}$, and two labeled constructions: abstraction and application. The labeling is done with labels taken from a set of *position labels* $\mathcal{L}$. This set is the disjoint union of two sets: the set $\mathcal{N} = \mathbb{N} - \{0\}$ of *numeric labels*, and the set $\mathcal{S}$ of *symbolic labels*. Each of the three sets $\mathcal{N}$, $\mathcal{S}$, and $\mathcal{L}$, is totally ordered. Namely, $\mathcal{N}$ is ordered with the natural number ordering, that we shall write $<_{\mathcal{N}}$; $\mathcal{S}$ is ordered with a linear order that we write $<_{\mathcal{S}}$; and, $\mathcal{L}$ is ordered by the order $<_{\mathcal{L}}$ such that $<_{\mathcal{L}} = <_{\mathcal{N}}$ on $\mathcal{N}$, $<_{\mathcal{L}} = <_{\mathcal{S}}$ on $\mathcal{S}$, and $\forall(n, p) \in \mathcal{N} \times \mathcal{S},\ n <_{\mathcal{L}} p$. In other words, all numeric labels are less than all symbolic labels.

We will denote variables by x, y, labels in $\mathcal{L}$ by p, q, reserving m, n to numbers in $\mathcal{N}$, and λ-expressions by capitals.

We can define the syntax of λ-terms as:

$$\begin{array}{rcll} M & ::= & x & \text{(variables)}, \\ & | & \lambda_p x.M & \text{(abstractions)}, \\ & | & M \ \widehat{p} \ M & \text{(applications)}. \end{array}$$

We will say of a term $\lambda_p x.M$ that it *"abstracts x at p in M,"* and of the term $M \ \widehat{p} \ N$, that it *"applies M to N through p."*

It will often be convenient to break the atomicity of an abstraction or an application. In the abstraction $\lambda_p x.M$, the part $\lambda_p x$ will be called its *abstractor*, and M its *body*. In the application $M \ \widehat{p} \ N$, the part $\widehat{p} \ N$ will be called the *applicator*. By *entity*, we will mean either an abstractor or an applicator (in which case we speak of a *labeled entity*), or simply a variable.

2.1.1 Relative and absolute positions

Before we delve into the technicalities of reduction, let us give some intuition to justify this syntax.

Symbolic labels are what we referred to as "keywords" in the introduction. A useful way of thinking of these symbols is to see them as *channel names* used for process communication [12]. Here, a process is a λ-term, where *sending* is performed by applicators and *receiving* by abstractors. If an application is performed ("sends arguments") through two different channels p and q, then clearly there cannot be any ambiguity as far as which abstractor will "receive" them. Hence, these reductions ("communications") may be done in any order, with the same end result. However, if that situation arises with $p = q$, then clearly the order in which they are performed will matter. In this case, the rules will insure that reduction will respect the order specified syntactically. In other words, several arguments sent through the same channel are "buffered" in sequence.[2]

If numeric labels are always kept explicit, then the above view applies to them as well. Indeed, recall from the introduction that the free syntax of function application to several arguments at a time uses their positions as Cartesian projections; *e.g.*, $f\big(a_1, \ldots, a_n\big)$ may be seen as the more explicit $f\big(1 \Rightarrow a_1, \ldots, n \Rightarrow a_n\big)$. However, numeric labels do not quite behave like symbolic labels in that a number is always *implicitly* seen as the *first* position *relatively* to the form on its left. More precisely, currying works by seeing each argument as the first one relatively to the form on its left. This has the benefit of simplifying the rule of functional reduction to be a *local* rule never needing to consider more than a single argument at a time. So, clearly, we do want to allow using relative argument positions.

Nevertheless, it is more natural to use absolute positions "packaged" as labeled Cartesian tuples. For instance, it is easier to write $\big(\lambda\big(1 \Rightarrow x, 2 \Rightarrow y, 4 \Rightarrow z\big).M\big) \frown \big(1 \Rightarrow a, 4 \Rightarrow b\big)$ rather than $\big(\lambda_1 x.\lambda_1 y.\lambda_2 z.M\big) \ \widehat{1} \ a \ \widehat{3} \ b$. However, the latter fully curried form is needed to express reduction with local rules. Fortunately, translation from the notation with absolute labels to a fully curried one with relative labels is in fact systematic: one need simply subtract from each

[2] In fact, we are also considering a possible variation of our calculus where this sequential buffering is not guaranteed. Rather, several arguments received on a given channel are chosen non-deterministically. This interesting twist yields essentially the functionality of asynchronous process communication, at the expense, of course, of confluence. That work is the object of our current study and will be reported later [2].

numeric label the number of numeric-labeled components to its left in the labeled Cartesian product. Namely, for any $i_k, j_\ell \in \mathcal{N}$ such that $i_k < i_{k+1}, j_\ell < j_{\ell+1}$, $\big(1 \leq k \leq n, 1 \leq \ell \leq m\big)$:

$$\begin{array}{l}\big(\lambda\big(i_1 \Rightarrow x_1, \ldots,\\ \quad i_k \Rightarrow x_k, \ldots,\\ \quad\quad i_n \Rightarrow x_n\big) . M\big) \frown \big(j_1 \Rightarrow N_1, \ldots,\\ \qquad\qquad\qquad\qquad j_\ell \Rightarrow N_\ell, \ldots,\\ \qquad\qquad\qquad\qquad\quad j_m \Rightarrow N_m\big)\end{array}$$

translates into:

$$\begin{array}{l}\big(\lambda_{i_1} x_1 \ldots\\ \quad \lambda_{i_k - k+1} x_k \ldots\\ \qquad \lambda_{i_n - n+1} x_n . M\big) \mathbin{\widehat{}}_{j_1} N_1 \ldots\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathbin{\widehat{}}_{j_\ell - \ell+1} N_\ell \ldots\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbin{\widehat{}}_{j_m - m+1} N_m.\end{array}$$

With this, we are justified to limit our syntax to that of relative-labeling lending itself to simpler local reduction rules, while still keeping the freedom of a flexible surface syntax with Cartesian tuples using absolute position labeling.

Now, a reasonable question that one may have is whether we could not also treat symbolic labels as we do numeric labels. That is, we could envisage using a function associating each symbol to its predecessor in the linear order of symbols, thus doing away with names altogether.[3] This, however, would be possible only if the order on $\mathcal{S}$ were not dense. Since, in practice, $\mathcal{L}$ is the free monoid, generated by a subset of the ASCII alphabet, and is densely ordered by lexicographic ordering, this is ruled out. Hence, symbolic labels always designate *absolute* positions of arguments. In other words, packaging symbolic-labeled arguments in labeled Cartesian tuples is always safe since they are not concerned with relative positioning. In fact, the ordering on symbols is only necessary as a trick to avert non-termination so that rules may perform well-founded label commutation.

2.1.2 Substitutions

Substitution of variables by λ-expressions needs the same precautions as in λ-calculus and obeys exactly the same rules. As usual, we use the equal sign $(=)$ to mean syntactic equality modulo α-conversion, defining α-conversion as for classical λ-calculus.

Let $\mathrm{FV}\big(M\big)$ be the set of free variables in M; that is, variables that are not abstracted anywhere in M. The expression $[N/x]M$ denotes the term obtained by replacing all the free occurrences of variable x by N in (an appropriate α-renaming of) M. That is,

[3] This would amount to "compiling them away" as alluded to in the introduction.

$$
\begin{array}{lcll}
[N/x]x & = & N & \\
[N/x]y & = & y & \text{if } y \ , \ y = x \\
[N/x]\big(M_1 \ \widehat{p} \ M_2\big) & = & \big([N/x]M_1\big) \ \widehat{p} \ \big([N/x]M_2\big) & \\
[N/x]\big(\lambda_p x.M\big) & = & \lambda_p x.M & \\
[N/x]\big(\lambda_p y.M\big) & = & \lambda_p y.[N/x]M & \\
 & & \text{if } y = x \text{ and } y \ \mathrm{FV}(N) & \\
[N/x]\big(\lambda_p y.M\big) & = & \lambda_p z.[N/x][z/y]M & \\
 & & \text{if } y = x \text{ and } y \ \mathrm{FV}(N), & \\
 & & \text{and } z \ \mathrm{FV}(N) \ \mathrm{FV}(M). &
\end{array}
$$

2.2 Reduction systems

We introduce three distinct groups of reduction rules: β-reduction and two reordering systems. What we shall eventually call label-selective λ-calculus is the system freely combining β-reduction and reordering.

2.2.1 β-Reduction

Intuitively, β-reduction for labeled terms can be performed as soon as an abstraction at position p is applied through the same position p to a term.

$$(\beta) \quad \big(\lambda_p x.M\big) \ \widehat{p} \ N \rightarrow [N/x]M$$

2.2.2 Reordering rules

Clearly, some reordering of abstractors and applicators must be performed in order to make β-reduction possible. There are two sets or reordering rules to consider: those dealing with at least one symbolic label and those dealing only with numeric labels. The difference, as explained above, lies in the fact that symbolic labels, being always explicit, can commute freely with others, symbolic or numeric, as long as they are distinct. On the other hand, numeric labels need to be kept in relative coherence so that they are implicitly always the first argument of the form on their left.

Symbolic labels

There are three rules to consider for swapping two adjacent labeled entities p and q when at least one among p or q is a symbolic label. Rule (1) commutes order of abstractors; Rule (2) commutes order of applicators; and Rule (3) moves an applicator into the body of an abstraction when the expression looks "almost" like a β-redex, were it not for $p \neq q$.

$$
\begin{array}{lll}
(1) & \lambda_p x.\lambda_q y.M \rightarrow \lambda_q y.\lambda_p x.M & \big(\text{if } p > q\big) \\
(2) & M \ \widehat{p} \ N \ \widehat{q} \ P \rightarrow M \ \widehat{q} \ P \ \widehat{p} \ N & \big(\text{if } p > q\big) \\
(3) & \big(\lambda_p x.M\big) \ \widehat{q} \ N \rightarrow \lambda_p x.\big(M \ \widehat{q} \ N\big) & \big(\text{if } p \neq q\big)
\end{array}
$$

Rule (3) must be performed modulo appropriate α-renaming in order to avoid capture.

Numeric labels

When two adjacent labeled entities are both numeric, the three above cases must be considered as well. Similar commutations can also take place, except that swapping must preserve relative coherence of implicit positions. This is simply done by decrementing the greater of the two positions.

Let m and n be two positive integers.

$$\begin{array}{lll}
(4) & \lambda_m x.\lambda_n y.M \to \lambda_n y.\lambda_{m-1} x.M & (\text{if } m > n) \\
(5) & M\ \widehat{m}\ N\ \widehat{n}\ P \to M\ \widehat{n}\ P\ \widehat{m-1}\ N & (\text{if } m > n) \\
(6) & (\lambda_m x.M)\ \widehat{n}\ N \to \lambda_{m-1} x.(M\ \widehat{n}\ N) & (\text{if } m > n) \\
(7) & (\lambda_m x.M)\ \widehat{n}\ N \to \lambda_m x.(M\ \widehat{n-1}\ N) & (\text{if } m < n)
\end{array}$$

Rules (6) and (7) must be performed modulo appropriate α-renaming in order to avoid capture. Of course, there is direct correspondence between the sets of rules for symbolic and numeric labels. Rules (1) and (2) are directly translated into (4) and (5), with the appropriate changes in labels. Rule (3) must be split into (6) and (7) in order to distinguish cases where $m < n$ and $m > n$.

2.2.3 Examples of reductions

Symbolic labels

We suppose that $p < q < r < s$,

$$\begin{array}{ll}
 & (\lambda_p x.\lambda_q y.\lambda_r z.M)\ \widehat{r}\ N\ \widehat{s}\ P\ \widehat{p}\ Q \\
\to_3 & (\lambda_p x.((\lambda_q y.\lambda_r z.M)\ \widehat{r}\ N))\ \widehat{s}\ P\ \widehat{p}\ Q \\
\to_2 & (\lambda_p x.((\lambda_q y.\lambda_r z.M)\ \widehat{r}\ N))\ \widehat{p}\ Q\ \widehat{s}\ P \\
\to_\beta & (\lambda_q y.\lambda_r z.[Q/x]M)\ \widehat{r}\ [Q/x]N\ \widehat{s}\ P \\
\to_3 & (\lambda_q y.((\lambda_r z.[Q/x]M)\ \widehat{r}\ [Q/x]N))\ \widehat{s}\ P \\
\to_\beta & (\lambda_q y.[Q/x][N/z]M)\ \widehat{s}\ P \\
\to_3 & \lambda_q y.([Q/x][N/z]M)\ \widehat{s}\ P
\end{array}$$

Numeric labels

$$\begin{array}{ll}
 & (\lambda_2 x.\lambda_1 y.\lambda_2 z.M)\ \widehat{4}\ N\ \widehat{5}\ P\ \widehat{2}\ Q \\
\to_4 & (\lambda_1 y.\lambda_1 x.\lambda_2 z.M)\ \widehat{4}\ N\ \widehat{5}\ P\ \widehat{2}\ Q \\
\to_7 & (\lambda_1 y.((\lambda_1 x.\lambda_2 z.M)\ \widehat{3}\ N))\ \widehat{5}\ P\ \widehat{2}\ Q \\
\to_5 & (\lambda_1 y.((\lambda_1 x.\lambda_2 z.M)\ \widehat{3}\ N))\ \widehat{2}\ Q\ \widehat{4}\ P \\
\to_7 & (\lambda_1 y.\lambda_1 x.((\lambda_2 z.M)\ \widehat{2}\ N))\ \widehat{2}\ Q\ \widehat{4}\ P \\
\to_\beta & (\lambda_1 y.\lambda_1 x.[N/z]M)\ \widehat{2}\ Q\ \widehat{4}\ P \\
\to_7 & (\lambda_1 y.((\lambda_1 x.[N/z]M)\ \widehat{1}\ Q))\ \widehat{4}\ P \\
\to_\beta & (\lambda_1 y.[Q/x][N/z]M)\ \widehat{4}\ P \\
\to_7 & \lambda_1 y.([Q/x][N/z]M\ \widehat{3}\ P)
\end{array}$$

3 Proof of confluence

3.1 Pseudo-reduction

3.1.1 Rules

Pseudo-reduction rules are intended to make reordering systems confluent in the absence of β-reduction. They promote the formation of new reordering redexes by commutation over β-redexes. The idea is that, without β-reduction, β-redexes just sit there, presenting "obstacles" to the formation of reordering redexes. Hence, we need pseudo-reduction rules to simulate the promotion of reordering redexes that would appear if the β-reduction had been performed. We simulate that effect by having a labeled entity "jump" into, or out of, the body of the abstraction part of a β-redex through its "λ-membrane" and seek reordering on the other side of that membrane. There are two cases: (a) one corresponding to having an applicator jump "into" the body of the abstraction part of a β-redex, and (b) the other corresponding to having an abstractor jump "out of" it. Namely, for *any* labels p, q:

$$(a)\quad (\lambda_p x.M)\ \widehat{}_p\ N\ \widehat{}_q\ P \to (\lambda_p x.(M\ \widehat{}_q\ P))\ \widehat{}_p\ N$$
$$(b)\quad (\lambda_p x.\lambda_q y.M)\ \widehat{}_p\ N \to \lambda_q y.((\lambda_p x.M)\ \widehat{}_p\ N)$$

These two rules must be performed modulo appropriate α-renaming in order to avoid capture. Note that neither rule actually destroys the occurrence of the β-redex which stays there.

Example 3.1 If we use only reordering rules, $(\lambda_1 x.\lambda_2 y.x\ \widehat{}_1\ y)\ \widehat{}_2\ a\ \widehat{}_1\ b$ can be reduced by Rules (7) and (6) yielding $A = (\lambda_1 x.\lambda_1 y.x\ \widehat{}_1\ y\ \widehat{}_1\ a)\ \widehat{}_1\ b$. It can also be reduced by Rule (5) to $B = (\lambda_1 x.\lambda_2 y.x\ \widehat{}_1\ y)\ \widehat{}_1\ b\ \widehat{}_1\ a$. Both terms A and B are normal forms with respect to reordering. We need pseudo-reduction to recover confluence. Namely, applying Rule (a) to B promotes the appearance of a redex for Rule (6), which yields A.

We will *not* use pseudo-reduction as a reduction system. Rather, we use it to define an equivalence relation modulo which confluence will be defined for partial reduction systems that we are to consider.

3.1.2 Pseudo-reduced form equivalence

Since some critical pairs appear between pseudo-reduction and reordering rules, we introduce an equivalence relation that combines their effects. We define two pseudo-reduced form (PRF) equivalences: (I) and (II). PRF equivalence (I) inverts the nesting of two β-redexes, with the necessary precautions. Namely, if $x \notin \mathrm{FV}(N)$, and with appropriate renaming:

$$(\mathrm{I})\quad (\lambda_p x.((\lambda_q y.M)\ \widehat{}_q\ N))\ \widehat{}_p\ P \to (\lambda_q y.((\lambda_p x.M)\ \widehat{}_p\ P))\ \widehat{}_q\ N$$

This defines an equivalence by transitive closure, since applying twice this rule at the same occurrences is the identity. Note that it does not bother requiring that $y \notin \mathrm{FV}(P)$. Indeed, this condition is always satisfied because P is external in the original form, and had it contained y, it would have been renamed to make the terms distinct.

PRF equivalence (II) is a renaming of labeling for any β-redex. Namely,

$(\mathrm{II})\quad (\lambda_p x.M)\ \widehat{p}\ N \leftrightarrow (\lambda_q x.M)\ \widehat{q}\ N$

This last equivalence will be essential to identify normal forms for reordering that differ only by the labeling of some β-redexes. Such situations would not occur if only symbolic labels were considered. However, it can arise for numeric labels that may, or may not, have been decremented through reordering.

3.2 Combined systems and restricted reductions

Using the rules above, there are in fact several reductions systems that we can consider depending on various combinations of the groups of rules. The full system that we will aim for is the free combination of β-reduction and reordering rules (System 2.2.1 + 2.2.2). We will call it *selective λ-calculus*. However, we will first consider the following partial reduction systems.

We will first focus on reordering rules alone, and prove a few useful properties. To do this we have to add some rules to preserve confluence: pseudo-reductions (a) and (b). So the system we are really interested in is System 2.2.2 + 3.1.1. We will still call it the *reordering system*. An *order-normal form* is a normal form for the reordering system. For this reordering system we will consider a special class of reductions, called *standard reorderings*, which are innermost reorderings.

We will also consider a partial reduction system called *β-reordering*. It is β-reduction on order-normal forms, where the result of each step is normalized by the reordering system.

The last system, or *label-parallel system*, which makes the link between selective λ-calculus and β-reordering, is the combination of all three reduction systems (System 2.2.1 + 2.2.2 + 3.1.1).

For each of these reduction systems, we shall use the symbol $\rightarrow$ to indicate a single reduction step using any of system's rules, and $\rightarrow_r$ if the rule uses Rule (r). When unconcerned by termination, we shall accept the (possibly subscripted) step $(M \rightarrow M)$ in this relation. As usual, $\stackrel{*}{\rightarrow}$ is the reflexive and transitive closure, also possibly subscripted. Given a reduction strategy ϱ, we will use the symbol $\triangleright_\varrho$ to denote the subrelation of $\stackrel{*}{\rightarrow}$ using only ϱ-reduction steps. For example, $\triangleright_{std}$ for standard reorderings, $\triangleright_{mcd}$ for minimal complete developments, *etc.*

Confluence modulo an equivalence relation is defined as follows ([8]):

Definition 1 *A relation* $\rightarrow$ *is* confluent modulo *an equivalence* $\sim$ *iff*

$\forall x, y, x', y' \quad x \sim y \quad$ *and* $\ x \stackrel{*}{\rightarrow} x' \quad$ *and* $\ y \stackrel{*}{\rightarrow} y'$
implies
$\exists x'', y'' \qquad x' \stackrel{*}{\rightarrow} x''$ *and* $y' \stackrel{*}{\rightarrow} y''$ *and* $x'' \sim y''$.

Confluence will always mean at least modulo α-conversion. Furthermore, for several of our reduction systems, confluence will also be considered modulo *pseudo-reduced form equivalence* (PRF), that is the composition of the two equivalences *PRF* (I) and *PRF* (II) of 3.1.2.

3.3 Confluence of the reordering system

We can see a λ-term as a tree. An abstraction $\lambda_p x.M$ corresponds to a node labeled with the abstractor $\lambda_p x$ and it has one son corresponding to the body M. An application $M \ \widehat{p}\ N$ corresponds to a node labeled $\widehat{p}$ having two sons: a left one corresponding to M and a right one corresponding to N. A leaf node corresponds to a variable. Given a λ-term M, we will be interested in a particular sequence of entities of M, called its *spine*. Roughly, a term's spine is what is left of the tree after clipping all its right branches leaving only stubs of depth one. More precisely, the spine of a term M is the sequence of its entities, starting at the root, of the *rightmost* depth-first traversal of the tree obtained from the term's tree by replacing all the right expressions of applications by new variable names, each uniquely identifying the omitted subterm.

Definition 2 (Spine) *The spine of an expression is the sequence of its entities obtained inductively as follows:*

$$\begin{array}{lll} spine\,(\lambda_p x.M) & = (\lambda_p x), spine\,(M) & \\ spine\,(M\ \widehat{p}\ N) & = (\ \widehat{p}\ z_N), spine\,(M) & \text{where } z_N \text{ is new} \\ spine\,(x) & = (x) & \text{if } x \in \mathcal{V}. \end{array}$$

Example 3.2 The entity sequence making up the spine of the λ-term:

$$M \ = \ \lambda_p x.((\lambda_q y.(y\ \widehat{r}\ (\lambda_s u.u)))\ \widehat{p}\ (x\ \widehat{t}\ v))$$

is given by:

$$spine\,(M) \ = \ (\lambda_p x), (\ \widehat{p}\ z_1), (\lambda_q y), (\ \widehat{r}\ z_2), (y),$$

where z_1 and z_2 are new variables standing respectively for M's subterms $x\ \widehat{t}\ v$ and $\lambda_s u.u$.

Note that the spine of a λ-term's subterm is *not* necessarily a subsequence of the term's own spine. Given a term M, by *set of spines* of M, we mean the set of spines of all its subterms (including itself).

The *size* of a spine is the number of entities it contains. We shall speak of the *index* of an entity in a spine as its position in the spine, starting from the root. The deeper a subterm occurs in a term, the higher its index in that term's spine will be.

Lemma 1 (Stability of entities) *The reordering rules* (1)–(7) *do not produce any labeled entities nor do they destroy any. Moreover, a labeled entity stays on the same spine after any reordering rule application.*

Proof: A quick look at the rules shows that none moves an entity from a spine in the set of spines of a term to another one. Moreover, it is even possible to track entities through the transformations, considering that those entities corresponding to the same label occurrence on the two sides of the rule are in fact identical up to variable renaming details (by "same label occurrence" we include m and m 1, n and n 1). ∎

Innermost reordering of a term is performed following the following strategy.

Definition 3 (Standard reordering) *A standard reordering is a reordering which rewrites entities of lower spine index first.*

Definition 4 (Locally dependent entities) *An abstractor and an applicator are said to be locally dependent in a term whenever they constitute a β-redex. They are locally independent if they do not.*

Definition 5 (Dependent entities) *An abstractor and an applicator are said to be dependent in a term if they can become* locally dependent *by a standard reordering. Otherwise they are independent.*

Note that a consequence of Lemma 1 is that two dependent entities are always on the same spine.

Lemma 2 *An abstractor cannot cross (exchange indexes with) an applicator of higher index in the spine. Or, equivalently, an applicator cannot cross an abstractor of lower index.*

We call a *masked step* during reordering an application of (a) replaced by two applications of System 2.2.2—namely, (2) then (3), or (5) then (7)—or the similar case for (b). Entities on different spines being independent, a step is masked only relatively to one spine, and disappears in the next reduction along its spine.

Lemma 3 *During a standard reordering locally dependent entities stay locally dependent, except temporarily during* masked steps *along their spine.*

Proof: Lemma 2 shows that any rewriting involving a local entity dependence can be done by (a) or (b), which does not separate the two entities. When $p > q$, (a) can be replaced by application of another rule but then the next step along that spine necessarily puts them together again, since no left subexpression of the reduced one is rewritable.

There are cases too where application of (b) is replaced by two applications of 2.2.2, because of standard reordering constraints, but the same phenomenon takes place: (1) (3) or (4) (7). ∎

Theorem 1 (Termination of standard reordering) *Standard reordering is Noetherian.*

Proof: Since there is no interaction between different spines in this system, it is sufficient to show that it is Noetherian along one spine.

In fact, we show that, for each spine, standard reordering terminates in $O(n^2)$ operations, n being the size of the spine. Moreover, when we take a minimal reducible subexpression, only the first step is on the whole subexpression, next ones being on a smaller one.

Base cases: $n = 0, 1$. No rule to apply.

Induction steps:

application of a rule from 2.2.2: if there is no possible reduction in the resulting subexpression, done: we added only one operation.

If there is a possible reduction, it is in a shorter subexpression: no rule is immediately reversible. In particular, a situation where (a) or (b) is applicable to the subexpression cannot be obtained

from a single reduction on a minimal subexpression. It would mean for (a) that $\widehat{p}$ and $\widehat{q}$ commuted, but then $p = q$ and a standard reordering would have permuted λ_p and $\widehat{q}$ first. For (b), λ_p and $\widehat{p}$ should have commuted, which is impossible.

If the reduction in the shorter subexpression is from 2.2.2, then the two first entities are in their original order (*i.e.*, it was order-normal), with the same order on their labels(for numbers): if it was an abstraction that went through (4) their labels did not change, and if it was an abstraction, its label was lower than the two others $(5, 6)$ and each is reduced by 1, or it was higher than the two (7) and they did not change, or higher than the first (7) and lower or equal to the second but then they become at worst equal and still no reduction is possible. So, we can go on by induction.

Otherwise, if the reduction in the shorter subexpression uses (a), then we obtain the three first entities in their original order (*i.e.*, normal here too), and their labels unchanged, except if Rule (6) has been used, but there is no new reduction possible since we have Lemma 2. So we can go on by induction on an even shorter subexpression.

And last, if (b) is used, we have certainly $q \ p$ or (1) would apply. If (7) has been used before, then $m \ p$, so that $m \ q$ and there is no possible reduction between those two abstraction. If another rule was applied before, two cases. 1) the first entity is an abstraction, Rule (3) or (6) was applied, and since it was next to λ_p its label is still strictly inferior to q. 2) the first entity is an application, its label was p, and no rule permit this configuration. Here too we can go on by induction.

application of Rule (a): we know then that there is no locally independent abstraction in M, otherwise one of rules (b), (1), (3), (4), (6) or (7) could be applied in a subexpression. It means that by Lemmas 2 and 3 all subsequent reductions will be in $(e \ \widehat{q} \ e_2)$. Induction hypothesis is applicable.

application of Rule (b): either M starts with a locally independent abstraction, with a label greater or equal to q, which is greater or equal to p (otherwise Rules (1) or (4) would apply), and the next step is (b) again: its label being greater or equal to q we can go on by recurrence; either M starts with an application, and the full subexpression is order-normal.

In all the case we have shown that we can go on by induction and that the introduction (at its root) of a new entity in a subexpression causes at most n operations, n being the size of the subexpression. We can then conclude that standard reordering terminates in less than $\frac{n(n-1)}{2}$ steps, which is $O(n^2)$. ∎

Theorem 2 (Confluence of standard reordering) *Standard reordering is confluent modulo PRF equivalence.*

Proof: We examine confluence in each spine, and will first show local confluence.

In each spine there can possibly be at most one minimal subexpression to reduce, by definition of the standard reordering.

By the structure of the patterns used in rules, only the beginning of the subexpression determines the choice of the rule. Rules in 2.2.2 are determined by the two first entities, and are completely distinct by the difference application/abstraction and the order of the labels. Rule (a) may apply when a rule of 2.2.2 is possible too, that is when $p > q$. Since we consider the system modulo PRF equivalence, that means at any time, by second equivalence. In case (2) is applied, the next is necessarily (1), and the result is the same. In case (5) is applied, the next is (4), and we still obtain the same result modulo PRF equivalence.

For the same reasons, Rule (b) may be replaced by rules from 2.2.2, with same conclusion.

We do not have to consider the first equivalence in detail: since we are working on a spine, and modulo second equivalence too, all locally dependent pairs look the same.

So in each spine the next possible step is determined, or it just may be divided in two steps, which does not matter for confluence, since no other step can delay the second one. This determinism proves the local confluence of this system modulo PRF equivalence.

By a theorem in [8], a system Noetherian and locally confluent modulo a relation is confluent modulo this relation. ∎

Proposition 1 *Standard order-normal form and order-normal form are equal modulo PRF equivalence.*

Proof: No reordering rule may apply on a standard order-normal form. A reordering rule may be impossible in a standard reordering only when another reordering rule is possible. ∎

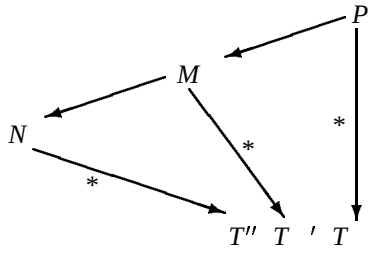
Theorem 3 *The reordering system is confluent modulo PRF equivalence.*

Proof: It is sufficient to show the confluence in each spine.

We show that for each rule that might be applied, a standard reordering of the minimal subexpression containing it would give the same result modulo PRF as taking the result of the rule and applying it a standard reordering. That is:

$$P\ M\ )\ (T, T\quad{}')\ P \triangleright_{std} T, M \triangleright_{std} T', T\ \ T\ \ {}'.$$

It is enough, since it shows that for any expression having an order-normal form (and they all have), applying any reordering rule will not change this order-normal form modulo PRF equivalence, so that applying different reorderings to an expression will always be reducible to the same order-normal form, in a finite number of steps, since standard reordering is Noetherian.



We prove it by induction on the length of the longest standard reordering.

- (a) If the standard order-normal form starts with an application, then (a) (or an equivalent two steps form as above) is the standard operation. Otherwise $(\lambda_p x.M)\ \widehat{}_p\ N$ would first go in by successive applications of (b) or its equivalent, and then $\widehat{}_q\ P$ would go in. Since (b) does not modify its context, if $\widehat{}_q\ P$ is not dependent, then the same Rule (a) would be applied later in the standard reordering and we would finally obtain the same order-normal form. If it is dependent, then the order of the two pairs will be inverted, but confluence is preserved by PRF equivalence.
- (b) If the standard order-normal form of M starts with an application, then (b) or its equivalent two step operation is the standard operation. Otherwise $\lambda_q y$ may first go in, but will eventually be passed by the p pair using Rule (b) or equivalent. The result is identical modulo PRF (II).

(1) This is the standard rule if the standard order-normal form of M starts with an abstraction, or an application greater or equal to q. Otherwise, p does not change and q can only be reduced (if numeric), so they would switch (exchange indexes) anyway and the order-normal form be the same.

(2) If the two are independent, or only p dependent then the switch would happen later, but the result be the same. If only q is dependent, then there would be no switch, but the result is still the same since only q may change if numeric. If the two are dependent, then the result is identical modulo PRF (I).

(3) This is the standard rule if the standard order-normal form of M starts with an abstraction, or an application greater or equal to p. Otherwise if $q < p$ and q is dependent, then in fact they would not have to switch, but the difference is masked by PRF (I). If it is not dependent, they would switch later.

(4) This is the standard rule if the standard order-normal form of M starts with an abstraction, or an application greater or equal to n. Otherwise $\lambda_n y$ would go in by Rule (4), which does not change context, then $\lambda_m x$, which would finally cross it with the same difference between labels.

(5) This is the standard rule if the standard order-normal form of M starts with an abstraction greater or equal to m. Else if the two are independent, in a standard reordering $\widehat{m}\, N$ would have first gone in and then $\widehat{n}\, P$ coming after crossing it in the independent application part. Making the switch first is not a problem, since $\widehat{m-1}\, N$ will then move in a context where every label greater than it has been decremented too, and $\widehat{n}\, P$ only stops after the switch.

If only $\widehat{n}\, P$ is dependent, then (5) was superfluous since we will finally have the original order. But the result will be the same. If only $\widehat{m}\, N$ is dependent, the switch is needed, but may have occurred later as an (a) rule. If the two are dependent, the results are identical up to PRF (I).

(6) This is the standard rule if the standard order-normal form of M starts with an abstraction, or an application greater or equal to m. Else, they would have to switch anyway. If $\widehat{n}\, N$ is independent, then the switch would be caused by the same Rule (6), otherwise $\lambda_m x$ would first cross the dependent abstraction, be reduced by (4), and the switch would cause no reduction as done by (b). So the result is the same up to PRF (II).

(7) This is the standard rule if the standard order-normal form of M starts with an abstraction, or an application greater or equal to m. Else they would have to switch anyway. Whether $\widehat{n}\, N$ is dependent or not does not matter: the switch will be always caused by Rule (7) since $m < n$.

∎

Lemma 4 *Locally dependent pairs stay locally dependent in the order-normal form.*

Proof: Lemma 3 and confluence modulo PRF equivalence. ∎

The following proposition is not really needed in the rest of the proof, but we give it for the sake of completeness.

Proposition 2 *The reordering system is Noetherian.*

Proof: The only cases of superfluous operation relatively to a standard reordering is when an dependent application crosses another application while going in. This is superfluous since it will stop in a situation where second equivalence applies, and reordering is not needed (if the second application is dependent), or twice superfluous if the second application is independent since it will have to cross it again.

However, these situations are limited to one for each pair of applications, which still gives a maximum reordering time in $O(n^2)$. ∎

3.4 Confluence of β-reordering

Definition 6 (β-Reordering) *A* β-reordering *step is a* β*-reduction step immediately followed by a reduction to order-normal form.*

Since reordering is Noetherian and confluent modulo PRF-equivalence, this completely defines a reduction rule on the quotient of order-normal λ-terms modulo PRF-equivalence. The one-step β-reordering relation is denoted as $\rightarrow_{\beta\downarrow}$.

Not to worry about renaming problems during reordering, we will assume that all free variables and abstraction variables have distinct names.

For Definition 6 to stand we have to prove that PRF-equivalent order-normal expressions are still equivalent after β-reordering. PRF equivalence (II) is not a problem: either the modified entity dependence is reduced, and then, the label does not matter, or it is not, and then it is changed back to what it was. As for, PRF equivalence (I), we can assume that the reduced dependent pair is always in the least possible index in the spine. If it is not, it does not matter since this equivalence obviously preserves entity dependencies.

This justifies us in the rest of this section, to consider no longer terms, but their equivalence classes modulo PRF equivalence.

For any label-selective λ-term M, we will note $M\downarrow$ its order-normal form. To lighten notation, but only in this section, we will write a term M when we actually mean $\downarrow$ implicitly. We shall do that everywhere in the section, except of course when we prove properties about it. That is, everywhere except Lemmas 5, 6 and 8. That means that generally $M = N$ should be read as $M\downarrow = N\downarrow$, $M \rightarrow_{\beta\downarrow} N$ as $M\downarrow \rightarrow_{\beta\downarrow} N\downarrow$, and $M \triangleright_{mcd} N$ as $M\downarrow \triangleright_{mcd} N\downarrow$.

From here on, the proof of β-reordering confluence follows the Martin-Löf-Tait scheme as in [7]. By *contracting* a β-redex, we mean applying the corresponding step of β-reduction.

Definition 7 (Residuals) *Let R, S be* β*-redexes in an order-normal* λ*-term P. When R is contracted, let P change to P'. The residuals of S with respect to R are redexes in P', defined as follows:*

- *R, S are* non-overlapping parts *of P. Then contracting R leaves S unchanged. This unchanged S in* P' *is called the residual of S.*
- $R = S$*. Then contracting R is the same as contracting S. We say S has no residual in* P'*.*
- R is part of S and $R \neq S$*. Then S has form* $(\lambda_p x.M)\ \widehat{p}\ N$ *and R is in M or in N. Contracting R changes M to* M' *or N to* N'*, and S to* $(\lambda_{p'} x.M')\ \widehat{p'}\ N$ *or* $(\lambda_p x.M)\ \widehat{p}\ N'$*; this is the residual of S.*
- S is part of R and $S \neq R$*. This case will not happen in our proof.*

Note that in this definition, the meaning of *non-overlapping* is taken in a large sense: it means that there is a configuration of entity dependencies such that R and S are not overlapping. Note also that in these three cases S has at most one residual.

Lemma 5 (Substitution) *Reordering before or after a substitution does not change the result.*

$$[N/x]M = [N/x]M\downarrow$$

Proof: We shall only consider whether ends of spines (last variable) will be substituted or not. In each spine where it is substituted, we can conclude by confluence of reordering (reordering the outer part of the spine and then introducing the end is equivalent to reordering directly the whole spine). In spines where it is not, there is no problem since they are left unmodified. ∎

Lemma 6 (Induction)
$$M \to_{\beta\downarrow} M' \Rightarrow \lambda_p x.M \to_{\beta\downarrow} \lambda_p x.M'$$
$$M \to_{\beta\downarrow} M' \Rightarrow M \mathbin{\widehat{p}} N \to_{\beta\downarrow} M' \mathbin{\widehat{p}} N$$
$$N \to_{\beta\downarrow} N' \Rightarrow M \mathbin{\widehat{p}} N \to_{\beta\downarrow} M \mathbin{\widehat{p}} N'$$

Proof:

1. $M = \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.M_1$, with M_1 an order-normal form starting with an abstraction and $p_i \;\; p_{\;\; i+1}$. So that $M' = \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.M_1'$, with M_1' any order-normal expression, and $M_1 \;\; {}_{\beta\downarrow} \; M_1'$. Hence
$$\begin{array}{rcl} \lambda_p x.M & = & \lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1 \\ & {}_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1' \\ & = & \lambda_p x.M' \end{array}$$
2. If $\mathbin{\widehat{p}} N$ is dependent then
$$\begin{array}{rcll} M \mathbin{\widehat{p}} N & = & (\lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1) \mathbin{\widehat{p}} N & M_1 \text{ as before}, p_i \;\; p_{\;\; i+1} \\ & = & \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.((\lambda_{p'} x.M_1) \mathbin{\widehat{p'}} N) & \text{(order normal form)} \\ & {}_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.((\lambda_{p'} x.M_1') \mathbin{\widehat{p'}} N) & \\ & = & (\lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1') \mathbin{\widehat{p}} N & M_1' \text{ order normal} \\ & = & M' \mathbin{\widehat{p}} N & \end{array}$$
If $\mathbin{\widehat{p}} N$ is not dependent then
$$\begin{array}{rcl} M \mathbin{\widehat{p}} N & = & (\lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.A(z \mathbin{\widehat{q_1}} N \ldots \mathbin{\widehat{q_m}} N_m)) \mathbin{\widehat{p}} N \\ & = & \lambda_{p_1} x_1 \ldots \lambda_{p_n'} x_n.A(z \mathbin{\widehat{q_1}} N \ldots \mathbin{\widehat{p'}} N \mathbin{\widehat{q_m'}} N_m) \\ & {}_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p_n'} x_n.A'(Z' \mathbin{\widehat{q_1}} N_1' \ldots \mathbin{\widehat{p'}} N \ldots \mathbin{\widehat{q_m'}} N_m') \\ & = & (\lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.A'(Z' \mathbin{\widehat{q_1}} N_1' \ldots \mathbin{\widehat{q_m}} N_m')) \mathbin{\widehat{p}} N \\ & = & M' \mathbin{\widehat{p}} N \end{array}$$
where A is the entity dependencies, A' the reduced dependencies, and N is unchanged because all its variables are free.
3. By independence of spines.

∎

Let $R_1, \ldots, R_n \; (n \geq 0)$ be redexes in a term P. An R_i is called *minimal* iff there is a configuration of entity dependencies in which it properly contains no other R_j. That is, there is a configuration of entity dependencies where it is the most internal redex and there is no redex in applications of spine index higher than or equal to its own.

A *minimal complete development* (MCD) of $\{R_1, \ldots, R_n\}$ in P is a sequence of contractions on P performed as follows:

- First, contract any minimal R_i (say $i = 1$ for convenience). This leaves at most $n - 1$ residual $R_2', \ldots, R_n'$, of $R_2, \ldots, R_n$.

- Then, contract any minimal R'_j. This leaves at most $n-2$ residuals.
- Repeat the above two steps until no residuals are left.

Note that this process is non-deterministic, and thus there are more than one such sequence of contractions.

Definition 8 (MCD) *Let P be a term as above, and Q a term. We write $P \triangleright_{mcd} Q$ iff Q is obtained from P by* minimal complete development *of the set $\{R_1, \ldots, R_n\}$.*

Note that if $M \triangleright_{mcd} M'$ and $N \triangleright_{mcd} N'$, then $M \;\widehat{p}\; N \triangleright_{mcd} M' \;\widehat{p}\; N'$. (*cf.*, Lemma 6)

Lemma 7 *If $M \triangleright_{mcd} M'$ and $N \triangleright_{mcd} N'$, then*

$$[N/x]M \triangleright_{mcd} [N'/x]M'.$$

Proof: We proceed by induction on M. Let $R_1, \ldots, R_n$ be the redexes developed in the given MCD of M.

1. $M = x$. Then n=0 and $M' = x$, so
$$[N/x]M = N \triangleright_{mcd} N' = [N'/x]M'.$$
2. x FV(M). Then x FV$(M \quad ')$, so
$$[N/x]M = M \triangleright_{mcd} M' = [N'/x]M'.$$
3. $M = \lambda_p y.M_1$. Then each β-redex in M is in M_1, so M' has form $\lambda_p y.M'_1$ where $M_1 \triangleright_{mcd} M'_1$. Hence
$$\begin{array}{lll} [N/x]M & = & [N/x](\lambda_p y.M_1) \quad \text{Lemma 5} \\ & = & \lambda_p y.[N/x]M_1 \quad \text{since } y \text{ FV}(xN) \\ & \triangleright_{mcd} & \lambda_p y.[N'/x]M'_1 \quad \text{by induction hypothesis} \\ & = & [N'/x]M' \quad \text{since } y \text{ FV}(xN \quad ') \end{array}$$
4. $M = M_1 \;\widehat{p}\; M_2$ and each R_i is in M_1 or M_2. Then M' has form $M'_1 \;\widehat{p}\; M'_2$ where $M_j \triangleright_{mcd} M'_j$ for $j = 1, 2$. Hence
$$\begin{array}{lll} [N/x]M & = & ([N/x]M_1) \;\widehat{p}\; ([N/x]M_2) \quad \text{Lemma 5} \\ & \triangleright_{mcd} & ([N'/x]M'_1) \;\widehat{p}\; ([N'/x]M'_2) \quad \text{by ind. and note above} \\ & = & [N'/x]M'. \end{array}$$
5. $M = (\lambda_p y.L) \;\widehat{p}\; Q$ and one R_i, say R_1, is M itself and is contracted last, and the others are in L or Q. (If it is not contracted last then we have $M = (\lambda_q z.K) \;\widehat{q}\; O$ too, and this one is contracted last). Hence the MCD has form
$$\begin{array}{lll} M = (\lambda_p y.L) \;\widehat{p}\; Q & \triangleright_{mcd} & (\lambda_p y.L') \;\widehat{p}\; Q' \quad (L \triangleright_{mcd} L', Q \triangleright_{mcd} Q') \\ & \beta\downarrow & [Q'/y]L' \\ & = & M'. \end{array}$$
By induction hypothesis we have MCD's of $[N/x]L$ and $[N/x]Q$. Hence

$$\begin{array}{rcll}
[N/x]M & = & (\lambda_p y.[N/x]L) \;\widehat{p}\; ([N/x]Q) & \text{since } y \text{ FV}(xN) \\
 & \triangleright_{mcd} & (\lambda_p y.[N'/x]L') \;\widehat{p}\; ([N'/x]Q') & \text{induction} \\
 & \beta\downarrow & [([N'/x]Q')/y][N'/x]L' & \\
 & = & [N'/x][Q'/y]L' & \\
 & = & [N'/x]M'. &
\end{array}$$

This reduction is an MCD, as required.

∎

Lemma 8 (Proof induction) *If there is an MCD*

$$\begin{array}{rcll}
P = (\lambda_p x.M) \;\widehat{p}\; N & \stackrel{*}{\rightarrow}_{\beta\downarrow} & (\lambda_p x.M') \;\widehat{p}\; N' & \\
 & \rightarrow_{\beta\downarrow} & [N'/x]M' & = Q \\
 & \stackrel{*}{\rightarrow}_{\beta\downarrow} & Q' &
\end{array}$$

then there is an MCD

$$\begin{array}{rcll}
P = (\lambda_p x.M) \;\widehat{p}\; N & \stackrel{*}{\rightarrow}_{\beta\downarrow} & (\lambda_p x.M'') \;\widehat{p}\; N'' & \\
 & \rightarrow_{\beta\downarrow} & [N''/x]M'' & = Q'
\end{array}$$

Proof: Since this is an MCD, new reductions do not apply on redexes created in the substitution, and Q' has form $[N''/x]M''$.

We should then just show that there are MCD's $M \triangleright_{mcd} M''$ and $N \triangleright_{mcd} N''$, which proves that $(\lambda_p x.M) \;\widehat{p}\; N \triangleright_{mcd} [N''/x]M$, by Lemma 7.

Each step of the original MCD after $[N'/x]M'$ only modifies either N' or M' at a time. So that we can write $M' \;\; \beta\downarrow \; M_1 \;\; \beta\downarrow \ldots \;\; \beta\downarrow \; M''$, and since it is an MCD, $M' \triangleright_{mcd} M''$. Similarly $N' \triangleright_{mcd} N''$. And all the reductions performed are on the external level, that is permutable with our reduction on p in an MCD. So that $M \triangleright_{mcd} M''$ and $N \triangleright_{mcd} N''$. ∎

Lemma 9 *If $P \triangleright_{mcd} A$ and $P \triangleright_{mcd} B$, then there exists T such that $A \triangleright_{mcd} T$ and $B \triangleright_{mcd} T$.*

Proof: By induction on P.

1. $P = x$. Then $A = B = P$. Choose $T = P$.
2. $P = \lambda_p x.P_1$. Then all β-redexes in P are in P_1, and

   $$A = \lambda_p x.A_1, \qquad B = \lambda_p x.B_1,$$

   where $P_1 \triangleright_{mcd} A_1$ and $P_1 \triangleright_{mcd} B_1$. By induction hypothesis there is a T_1 such that

   $$A_1 \triangleright_{mcd} T_1, \qquad B_1 \triangleright_{mcd} T_1.$$

   Choose $T = \lambda_p x.T_1$.
3. $P = P_1 \;\widehat{p}\; P_2$ and all the redexes developed in the MCD's are in P_1, P_2. Then the induction hypothesis gives us T_1, T_2, and we choose $T_1 \;\widehat{p}\; T_2$.
4. $P = (\lambda_p x.M) \;\widehat{p}\; N$ and just one of the given MCD's involves contracting P's residual; say it is $P \triangleright_{mcd} A$. Then, by Lemma 8, there is an MCD with form

$$\begin{array}{rcll} P & = & (\lambda_p x.M) \mathbin{\widehat{p}} N & \\ & \triangleright_{mcd} & (\lambda_p x.M') \mathbin{\widehat{p}} N' & (M \triangleright_{mcd} M',\ N \triangleright_{mcd} N') \\ & \beta\downarrow & [N'/x]M' & \\ & = & A. & \end{array}$$

And the other MCD has form

$$\begin{array}{rcll} P & = & (\lambda_p x.M) \mathbin{\widehat{p}} N & \\ & \triangleright_{mcd} & (\lambda_p x.M'') \mathbin{\widehat{p}} N'' & (M \triangleright_{mcd} M'',\ N \triangleright_{mcd} N'') \\ & = & B. & \end{array}$$

The induction hypothesis applied to M, N gives us M^+, N^+ such that

$$\begin{array}{ll} M' \triangleright_{mcd} M^+, & M'' \triangleright_{mcd} M^+; \\ N' \triangleright_{mcd} N^+, & N'' \triangleright_{mcd} N^+. \end{array}$$

Choose $T = [N^+/x]M^+$. Then there is an MCD from A to T, thus, by lemma 7

$$A = [N'/x]M' \triangleright_{mcd} [N^+/x]M^+.$$

And for B,

$$\begin{array}{rcll} B & = & (\lambda_p x.M'') \mathbin{\widehat{p}} N'' & \\ & \triangleright_{mcd} & (\lambda_p x.M^+) \mathbin{\widehat{p}} N^+ & \\ & \beta\downarrow & [N^+/x]M^+ & \end{array}$$

5. $P = (\lambda_p x.M) \mathbin{\widehat{p}} N$ and both the given MCD's contract P's residual. Then (Lemma 8) we can give these MCD's form

$$\begin{array}{rcll} P & = & (\lambda_p x.M) \mathbin{\widehat{p}} N & \\ & \triangleright_{mcd} & (\lambda_p x.M') \mathbin{\widehat{p}} N' & \\ & \beta\downarrow & [N'/x]M' & \\ & = & A, & \end{array} \qquad \begin{array}{rcll} P & = & (\lambda_p x.M) \mathbin{\widehat{p}} N & \\ & \triangleright_{mcd} & (\lambda_p x.M'') \mathbin{\widehat{p}} N'' & \\ & \beta\downarrow & [N''/x]M'' & \\ & = & B. & \end{array}$$

Apply the induction hypothesis to M and N in case 4, and choose $T = [N^+/x]M^+$. Then Lemma 7 gives the result, as above.

∎

Theorem 4 *β-reordering is confluent modulo PRF equivalence.*

$$P \xrightarrow{*}_{\beta\downarrow} M,\ P \xrightarrow{*}_{\beta\downarrow} N \Rightarrow \left(\exists T\right) M \xrightarrow{*}_{\beta\downarrow} T,\ N \xrightarrow{*}_{\beta\downarrow} T.$$

Proof: By induction on the length of the reduction from P to M, it is enough to prove

$$P \quad _{\beta\downarrow},\ P \ ^{*}{}_{\beta\downarrow}\ N\ )\ (T)\ M \qquad ^{*}{}_{\beta\downarrow}\ T,\ N \ ^{*}{}_{\beta\downarrow}\ T.$$

Since a single β-reordering step is an MCD, it is sufficient to have

$$P \triangleright_{mcd} M,\ P \ ^{*}{}_{\beta\downarrow}\ N\ )\ (T)\ M \qquad ^{*}{}_{\beta\downarrow}\ T, N \triangleright_{mcd} T.$$

which is shown by an induction on the number of β-steps from P to N. ∎

3.5 Confluence of selective λ-calculus

In this section, $\rightarrow$ (or $\rightarrow_\lambda$) denotes the union of β-reduction and ordering rules (label-selective λ-calculus), and $\rightarrow_\omega$ is the union of all rules (label-parallel system). We will now *no longer* consider terms modulo PRF equivalence, except in the β-reordering diamond of Figure 1.

Definition 9 (Normalized reduction) *For each label-parallel reduction* $M_0 \rightarrow_\omega M_1 \rightarrow_\omega \ldots \rightarrow_\omega M_n$ *we define its* normalized reduction $N_0 \rightarrow_{\beta\downarrow} N \rightarrow_{\beta\downarrow} \ldots \rightarrow_{\beta\downarrow} N_n$ *by taking for each* N_i *the order-normal form* $M_i\downarrow$.

Proposition 3 *Normalized reduction is a* β*-reordering.*

Proof: We should verify that we really obtain a β-reordering by this process.

We can first remark that, since we have Lemma 4, all β-redexes in M_i are still β-redexes in N_i.

If $M_i \; M_{i+1}$ is a reordering step, then $N_i = N_{i+1}$. Else, $M_i \; M_{i+1}$ is a β-step, and we should show $N_i \; {}_{\beta\downarrow} \; N_{i+1}$. From our remark, we have $N_i \; {}_{\beta\downarrow} \; N'_i$, reducing the same redex. We will in fact construct two parallel reorderings of M_i and M_{i+1}. First, a standard reordering of M_i, from $M_i^0 = M_i$ to $M_i^k = N_i$, in which will not appear the masked step described about local entity dependencies, and ${}_c$ is one reordering step, or two if there is a masked step in the process. With such a reordering, we have at each step $M_i^j \; M_i'^j$ by a β-step. Then we define a reordering of M_{i+1} going through all $M_i'^j$'s. If $M_i^j \; {}_c \; M_i^{j+1}$ is a single step, then it cannot separate two locally dependent entities. There are four cases to consider:

1. If it is external to the reduced redex, then we can do the same reduction $M_i'^j \; {}_c \; M_i'^{j+1}$.
2. If it is internal, the β-reduction may only substitute some variables, but the reduction can still be applied. $M_i'^j \; {}_c \; M_i'^{j+1}$.
3. If it was an (a) or (b) reordering step over the redex, then it is superfluous after reduction, $M_i'^j = M_i'^{j+1}$.
4. In the two step case, it is equivalent to an (a) or (b) reordering step, as shown above. $M_i'^j = M_i'^{j+1}$.

Finally we can go from $M_i'^k$ to N'_i by a standard reordering. By confluence it gives $N'_i = N_{i+1}$, and the normalized reduction is correctly constructed. ∎

Theorem 5 (Confluence of label-parallel reduction) *The label-parallel system is confluent. That is,*

$$P \stackrel{*}{\rightarrow}_\omega M,\; P \stackrel{*}{\rightarrow}_\omega N \;\Rightarrow\; \left(\exists T\right)\, M \stackrel{*}{\rightarrow}_\omega T,\; N \stackrel{*}{\rightarrow}_\omega T.$$

Proof: We have

$$\begin{array}{llll} P & {}_\omega \; M_1 & {}_\omega \ldots & {}_\omega \; M_m \;\; = M, \\ P & {}_\omega \; N & {}_\omega \ldots & {}_\omega \; N_n \;\; = N. \end{array}$$

So that we obtain normalized reductions

$$\begin{array}{llll} P' & {}_{\beta\downarrow} \; M'_1 & {}_{\beta\downarrow} \ldots & {}_{\beta\downarrow} \; M'_m, \\ P' & {}_{\beta\downarrow} \; N'_1 & {}_{\beta\downarrow} \ldots & {}_{\beta\downarrow} \; N'_n. \end{array}$$

And by confluence of β-reordering,

$$\begin{array}{llllll} M'_m & = R_0 & \beta\downarrow R_1 & \beta\downarrow \ldots & \beta\downarrow R_r & = T, \\ N'_n & = S_0 & \beta\downarrow S_1 & \beta\downarrow \ldots & \beta\downarrow S_s & = T. \end{array}$$

Since normalized (β-reordering) reductions are confluent, and all steps used here are in the label-parallel system, the label-parallel system is confluent modulo PRF equivalence.

We can then reduce all the β-redexes present in the resulting term, and obtain full confluence. All differences masked by PRF equivalence are contained in the redexes, and since in this last stage we do not use pseudo-reduction, we do not create new differences. ∎

Theorem 6 (Confluence of label-selective λ-calculus) *The label-selective λ-calculus is confluent. That is,*

$$P \xrightarrow{*} M,\ P \xrightarrow{*} N \ \Rightarrow\ \left(\exists T\right) M \xrightarrow{*} T,\ N \xrightarrow{*} T.$$

Proof: By Theorem 5,

$$\begin{array}{llllll} M & = R_0 & \omega\ R_1 & \omega \ldots & \omega\ R_r & = T', \\ N & = S_0 & \omega\ S_1 & \omega \ldots & \omega\ S_s & = T'. \end{array}$$

But the absence of pseudo-reduction rules makes it impossible to follow these paths. Each time we have a (a) or (b) reduction, we should have a β-reduction in place. So we just have to rewrite them, which makes superfluous some later steps or couples of steps, that we will just suppress. It may duplicate some later reduced redexes too. In this case we will have to duplicate these later steps too, but this is finite.

We will finally have two expressions, coming from a PRF equivalent of T' by β-reduction only. The number of β-reductions done may differ, but reducing all the redexes which were present in T' is enough. That is,

$$\begin{array}{llllll} M & R'_0 & R'_1 & \ldots & R'_r & \overset{*}{}_\beta\ T, \\ N & S'_0 & S'_1 & \ldots & S'_s & \overset{*}{}_\beta\ T. \end{array}$$

where $R_i \overset{*}{}_\beta R'_i$ (for some PRF equivalent form of R_i), and $S_i \overset{*}{}_\beta S'_i$ (*idem*). So that finally, $M \overset{*}{} T$ and $N \overset{*}{} T$. ∎

Figure 1 shows a schematic diagram of the process.

4 Extended systems

These proofs are valid in fact for a whole range of systems. The most immediate ones are the two systems in which only numeric or only symbolic labels are accepted. Intuitively they represent two different way of introducing some commutativity in λ-calculus. Selective λ-calculus is their sum. Namely, it is obtained as the calculus over the set of labels which is their disjoint sum: $\mathcal{L} = \mathcal{S} + \mathcal{N}$.

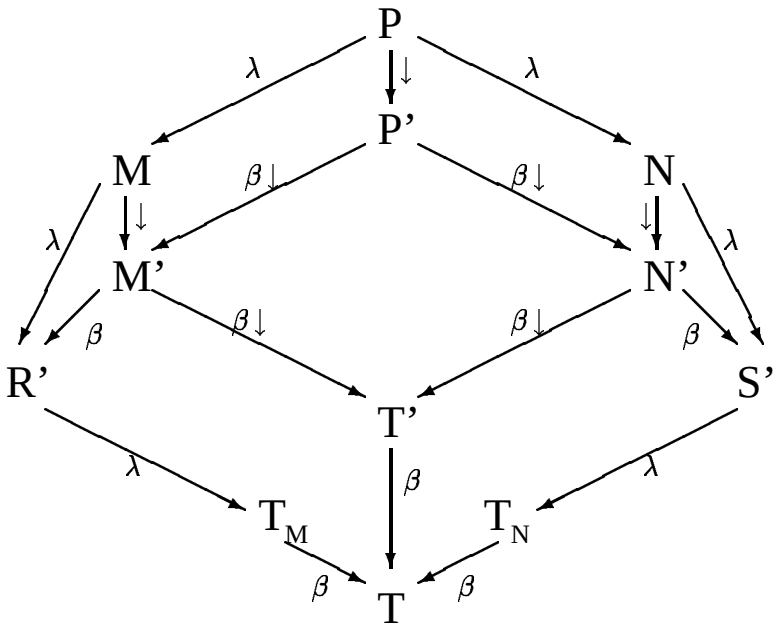


Figure 1: Schematic confluence of label-selective λ-calculus

4.1 Product system

Since we have a sum, it is natural to think of a product. That is, the calculus over the set of labels which is the Cartesian product of symbolic and numeric label. Namely, a label in now a pair symbol-numeral in $\mathcal{L} = \mathcal{S} \times \mathcal{N}$. This system corresponds to a calculus of named channels on which only numeric positions are considered. Per channel name, relative numeric position arguments can be done in that relative reordering of distinct positions is allowed. In a sense, it is orthogonal to the sum system, in that it separates names and numbers, although it provides each with the functionality of label-selective λ-reduction.

β-Reduction for product, as for the sum, requires identically labeled abstractor and applicator:

$$(\beta) \quad (\lambda_{pm} x.M) \ \widehat{pm} \ N \rightarrow [N/x]M$$

As for reordering, we need only slightly alter reordering rules of the sum system as shown in Figure 2. Clearly, all proofs done above on selective λ-calculus are still valid for this system: we take care of numbers only when symbols are identical, and have then the same rules.

4.2 Polynomial systems

If we have sums and products, we should be able to combine them. A *polynomial system* is defined by two mutually exclusive sets of names $\mathcal{S}$ and $\mathcal{T}$, and taking the set of labels to be $\mathcal{L} = \mathcal{S} \cup (\mathcal{T} \times \mathcal{N})$. The syntax is:

$$
\begin{array}{lll}
(1) & \lambda_{pm}x.\lambda_{qn}y.M \to \lambda_{qn}y.\lambda_{pm}x.M & p > q \\
(2) & M\ \widehat{}_{pm}\ N\ \widehat{}_{qn}\ P \to M\ \widehat{}_{qn}\ P\ \widehat{}_{pm}\ N & p > q \\
(3) & (\lambda_{pm}x.M)\ \widehat{}_{qn}\ N \to \lambda_{pm}x.(M\ \widehat{}_{qn}\ N) & p \neq q \\
(4) & \lambda_{pm}x.\lambda_{pn}y.M \to \lambda_{pn}y.\lambda_{p(m-1)}x.M & m > n \\
(5) & M\ \widehat{}_{pm}\ N\ \widehat{}_{pn}\ P \to M\ \widehat{}_{pn}\ P\ \widehat{}_{p(m-1)}\ N & m > n \\
(6) & (\lambda_{pm}x.M)\ \widehat{}_{pn}\ N \to \lambda_{p(m-1)}x.(M\ \widehat{}_{pn}\ N) & m > n \\
(7) & (\lambda_{pm}x.M)\ \widehat{}_{pn}\ N \to \lambda_{pm}x.(M\ \widehat{}_{p(n-1)}\ N) & m < n
\end{array}
$$

Figure 2: Reordering rules for polynomial system

$$M ::= x \mid \lambda_p x.M \mid \lambda_{qn}x.M \mid M\ \widehat{}_p\ M' \mid M\ \widehat{}_{qn}\ M'$$

with p in $\mathcal{S}$ and q in $\mathcal{T}$.

We just take Rules (1)–(3) of selective λ-calculus applied on $\mathcal{S} \cup (\mathcal{T} \times \mathcal{N})$ and Rules (4)–(7) of product systems, applied on $\mathcal{T}$.

Selective λ-calculus is a polynomial system where $\mathcal{T}$ is restricted to an unique constructor ϵ, abbreviated.

It is only for clarity that proofs where given for selective λ-calculus, rather than for any polynomial system, where they are anyway valid. We should always keep in mind that when we prove or build something on selective λ-calculus it will be nearly always generalizable to any polynomial system, for which we are just using an abbreviated notation.

5 Towards a transformation calculus

We will just give here a new notation, and explain in what way this notation suggests another extension of selective λ-calculus.

The intuition behind selective λ-calculus is no longer functions but functions over labeled arguments which behave like communicating processes through named or (relatively) numbered channels. Application corresponds to process communication. What we called entities are seen as *actions*: abstractors correspond to receiving and applicators to sending. But what about composition? It is easily defined for functions as $f \circ g = \lambda f.\lambda g.\lambda x.f(gx)$ in the classical calculus. We could of course think of a labeled composition one for each label. But this is rather weak. Particularly when we think of the powerful out-of-order currying power of our system.

Rather, we will just change notations, and define:

$$M\ \vee_p\ x \stackrel{def}{=} \lambda_p x.M$$

All terms now take the form $x \cdot P$, where P is a sequence of entities, or actions, and $\cdot$ the syntactic juxtaposition. Then we can obtain a more interesting composition with $M \circ P = M \cdot P$, where we do not specify anything about labels, and may have created more than one connection

at once. Here again P is a sequence of actions, and we would like to manipulate them as such. The potential of this notation is such that it simplifies considerably many proofs. For instance we could prove easily Böhm's expansion theorem using it. Since our extension is conservative, our proof is valid for classical λ-calculus as well.

Of course, all this starts to be strongly reminiscent of a calculus for process communication [12], although still a direct and conservative extension of classical λ-calculus. We believe that this is not coincidental. This idea is the object of our current research and we are actively exploring the deep connections with the various existing communication calculi. But, we shall say no more on this for now.

6 Conclusion and further work

Label-selective λ-calculus offers the advantage of realizing directly a more complete isomorphism of Cartesian products and function applications. An immediate consequence is a more convenient notation, and a more efficient, indeed concurrent, manner to extract arguments out of order.

Beyond the bare calculus, we have started studying a typed version of our calculus [6]. There, we propose a simply typed version of this calculus, and show that it extends to second order and polymorphic typing. For this last one there exists a most generic type, and we give the algorithm to find it.

A topic for further work along this idea is, of course compilation. As mentioned in the first section, we plan to extend the stack-based model of execution of λ-calculus with our label-selection scheme to realize efficient access to arguments regardless of position label. We have already adapted the calculus of explicit substitutions [1], as are currently working on a compiling scheme for label-selective λ-calculus based on it.

Also, we plan to study the work of Ohori [13] to elucidate the gains that this may have in the compilation of records. As for semantics, we have initiated work on a typed version of label-selective λ-calculus and a framework of models for it.

We have formulated and are studying several concurrent calculi extending label-selective λ-calculus towards full concurrency [2], including the provision for computable channel names. One of the gains expected is that λ-calculus will need not be *encoded* as in [11], but directly embedded as syntactic identity.

Finally, the real goal that has motivated our working out this calculus has been to use it for a useful generalization of object-oriented style of message passing. Method invocation based on the type of the first argument of a call can be elegantly explained by seeing a method definition in a class as a curried form with respect to the object instance of the class. Label-selective currying can thus reinstate the lost symmetry by distributing one partially-applied form for each arguments of the class of a method. As a result, message-passing can be used on any argument of a call, making labels act as channels. Our confluence result guarantees that the choice of channel does not matter. We plan to pursue this insight and investigate all its ramifications.

References

1. Martin Abadi, Luca Cardelli, and Jean-Jacques Lévy. Explicit substitutions. In *Proceedings of the Seventeenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (1990).

2. Hassan Aït-Kaci and Kathleen Milsted. Concurrent label-selective λ-calculus. PRL research report, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (forthcoming).

3. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. PRL Research Report 11, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (1991). (Revised, October 1992; to appear in the *Journal of Logic Programming*).

4. Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *Proceedings of TAPSOFT'89*, pages 149–161, Berlin, Germany (1989). Springer-Verlag. LNCS 351.

5. N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indag. Math.*, 34:381–392 (1972).

6. Jacques Garrigue and Hassan Aït-Kaci. Typed label-selective λ-calculus. PRL research report, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Mailmaison, France (forthcoming).

7. J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts.* Cambridge University Press, Cambridge, UK (1986).

8. Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821 (October 1980).

9. Peter J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4):308–320 (1965).

10. Henry Ledgard. *ADA: An Introduction, Ada Reference Manual (July 1980).* Springer-Verlag, New York, NY (1981).

11. Robin Milner. Functions as processes. Rapport de Recherche 1154, INRIA, Le Chesnay, France (February 1990).

12. Robin Milner. The polyadic π-calculus: A tutorial. LFCS Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, Edinburgh, UK (October 1991).

13. Atsushi Ohori. A compilation method fo ML-style polymorphic records. In *Proceedings of the Ninetheenth ACM Symposium on Principles of Programming Languages*, pages 154–165 (January 1992).

14. Guy L. Steele. *Common LISP: The Language.* Digital Press (1984).

PRL Research Reports

The following documents may be ordered by regular mail from:

Librarian – Research Reports
Digital Equipment Corporation
Paris Research Laboratory
85, avenue Victor Hugo
92563 Rueil-Malmaison Cedex
France.

It is also possible to obtain them by electronic mail. For more information, send a message whose subject line is `help` to `doc-server@prl.dec.com` or, from within Digital, to `decprl::doc-server`.

Research Report 1: *Incremental Computation of Planar Maps*. Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. May 1989.

Research Report 2: *BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic*. Bernard Serpette, Jean Vuillemin, and Jean-Claude Hervé. May 1989.

Research Report 3: *Introduction to Programmable Active Memories*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. June 1989.

Research Report 4: *Compiling Pattern Matching by Term Decomposition*. Laurence Puel and Ascánder Suárez. January 1990.

Research Report 5: *The WAM: A (Real) Tutorial*. Hassan Aït-Kaci. January 1990.†

Research Report 6: *Binary Periodic Synchronizing Sequences*. Marcin Skubiszewski. May 1991.

Research Report 7: *The Siphon: Managing Distant Replicated Repositories*. Francis J. Prusker and Edward P. Wobber. May 1991.

Research Report 8: *Constructive Logics. Part I: A Tutorial on Proof Systems and Typed λ-Calculi*. Jean Gallier. May 1991.

Research Report 9: *Constructive Logics. Part II: Linear Logic and Proof Nets*. Jean Gallier. May 1991.

Research Report 10: *Pattern Matching in Order-Sorted Languages*. Delia Kesner. May 1991.

---

† This report is no longer available from PRL. A revised version has now appeared as a book: "Hassan Aït-Kaci, *Warren's Abstract Machine: A Tutorial Reconstruction. MIT Press, Cambridge, MA (1991).*"

Research Report 11: *Towards a Meaning of LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, October 1992).

Research Report 12: *Residuation and Guarded Rules for Constraint Logic Programming*. Gert Smolka. June 1991.

Research Report 13: *Functions as Passive Constraints in LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, November 1992).

Research Report 14: *Automatic Motion Planning for Complex Articulated Bodies*. Jérôme Barraquand. June 1991.

Research Report 15: *A Hardware Implementation of Pure Esterel*. Gérard Berry. July 1991.

Research Report 16: *Contribution à la Résolution Numérique des Équations de Laplace et de la Chaleur*. Jean Vuillemin. February 1992.

Research Report 17: *Inferring Graphical Constraints with Rockit*. Solange Karsenty, James A. Landay, and Chris Weikart. March 1992.

Research Report 18: *Abstract Interpretation by Dynamic Partitioning*. François Bourdoncle. March 1992.

Research Report 19: *Measuring System Performance with Reprogrammable Hardware*. Mark Shand. August 1992.

Research Report 20: *A Feature Constraint System for Logic Programming with Entailment*. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. November 1992.

Research Report 21: *The Genericity Theorem and the Notion of Parametricity in the Polymorphic λ-calculus*. Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. December 1992.

Research Report 22: *Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite*. François Bourdoncle. January 1993.

Research Report 23: *Dessin à main levée et courbes de Bézier : comparaison des algorithmes de subdivision, modélisation des épaisseurs variables*. Thierry Pudet. January 1993.

Research Report 24: *Programmable Active Memories: a Performance Assessment*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. March 1993.

Research Report 25: *On Circuits and Numbers*. Jean Vuillemin. April 1993.

Research Report 26: *Numerical Valuation of High Dimensional Multivariate European Securities*. Jérôme Barraquand. March 1993.

Research Report 27: *A Database Interface for Complex Objects*. Marcel Holsheimer, Rolf A. de By, and Hassan Aït-Kaci. March 1993.

Research Report 28: *Feature Automata and Sets of Feature Trees*. Joachim Niehren and Andreas Podelski. March 1993.

Research Report 29: *Real Time Fitting of Pressure Brushstrokes*. Thierry Pudet. March 1993.

Research Report 30: *Rollit: An Application Builder*. Solange Karsenty and Chris Weikart. April 1993.

Research Report 31: *Label-Selective λ-Calculus*. Hassan Aït-Kaci and Jacques Garrigue. May 1993.

Research Report 32: *Order-Sorted Feature Theory Unification*. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. May 1993.

31

Label-Selective λ-Calculus

Hassan Aït-Kaci and Jacques Garrigue

digital

PARIS RESEARCH LABORATORY

85, Avenue Victor Hugo
92563 RUEIL MALMAISON CEDEX
FRANCE