Label-Selective λ-Calculus Syntax and Confluence

Hassan Aït-Kaci Jacques Garrigue

December 13, 1994

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 combines symbolic keywords with numeric positions. While the former enjoy free commutation, the latter and relative renumbering are needed to extend commutation to conflictuous names, and allow full currying. This extension of λ-calculus is conservative, in the sense that when we restrict ourselves to using only one label, 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.[1]

Keywords λ-Calculus, Record Calculus, Concurrency, Communication.

Note A short version of this article appeared in the *Proceedings of the 13th International Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, December 1993. This work was started while the first author was at Digital Equipment Corporation's Paris Research Laboratory, France, and the second author was at Université de Paris 7, France. At present, the authors may be contacted at the following addresses:

Hassan Aït-Kaci
School of Computing Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6, Canada
hak@cs.sfu.ca

Jacques Garrigue
Research Institute for Mathematical Sciences
Kyoto University
Kitashirakawa-Oiwakecho, Sakyo-ku
Kyoto 606-01, Japan
garrigue@kurims.kyoto-u.ac.jp

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 original manuscript.

[1] This is a reprint maintained for archive purposes; in this version, typos present in the published manuscript have been corrected.

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 [21, 17, 4]. 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).

On the other hand, when currying is supported, the situation has to be reduced to numeric positions alone, which are handled in an absolute 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 $\lambda x.\lambda y.f(y,x)$ 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:

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

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:

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

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 $1st$ and $2nd$ 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:[2]

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

as one uses for records, and thus that:

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

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 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 intuitive relation between our calculus and the notation with offsets introduced by de Bruijn [6] and used for the compilation of λ-calculus in the style of the SECD machine [16]. These offsets are used to denote

[2] Parse the following with '$\Rightarrow$' binding tighter than '$\times$' or '$\to$'.

the "physical" (topological) relation between a variable and its binder. Our labels denote associations between abstractions and applications. The same kind of shifting mechanic is used to keep the links during reductions. However those two kinds of indices work on two independent levels: variables and arguments. That means that we cannot easily encode selective λ-calculus into the calculus of explicit substitutions [1] for instance. The opposite is possible, thanks to the two-level structure of our calculus. Another connection is the possibility to combine them, obtaining two levels of indices. We have already adapted the calculus of explicit substitutions, and are currently working on a compiling scheme for label-selective λ-calculus based on it.

Another potential connection, albeit from an opposite viewpoint, is with the work of Ohori in compiling extensible records for functional programming [20]. 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. The techniques used to compile them, like index abstraction, may then be extended to our system. At this time, the connection is not formalized 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. Function application is not commutative. Moreover, it is implicitly left-associative in λ-calculus, by convention. Our idea is to reinstate the inherent concurrency lost by these syntactic limitations 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 [5, 19, 18].

In [5], 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 $\bar{\lambda}$. This idea is taken to its full extent by Robin Milner in [19] 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 [19, 5], but as explicit *position* names. This is a wholly different insight. In addition, the availability of *numeric* indices on 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 [3], 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 [18], but directly embedded as syntactic identity.

More recently, we became aware of the work of Laurent Dami [8, 9]. In his work, Dami develops a complex calculus of record objects with a part dealing with functions with named arguments. At a first glance, because it allows named arguments and out-of-order application, Dami's calculus looks reminiscent of the part of our calculus dealing with symbolic positions. It is in fact fundamentally different. In Dami's calculus, a functional abstraction is seen as a pair consisting of a name and an expression (its body)—the name (the "inlet" [*sic*]) designates the *result* of applying the function to arguments. Arguments are identified as the "unprotected" [*sic*] (*ie.* , free) names in the expression. To avoid confusion in name references, a "protection" level in the form of a string of backslashes may prefix a name's occurrence—if a name's occurrence is prefixed with n backslashes in an expression, that reference occurs nested n levels deep from its free scope.[3] Accordingly, an application is also a pair made of a name and an expression. The name designates the unprotected reference (the "outlet" [*sic*]) in the applied function's body to substitute for the expression. Hence, Dami's application is really instantiation through a first-order substitution that associates expressions to names.[4] Clearly, this view allows out-of-order argument consumption. In fact, Dami's functions are one-fielded records from which to extract the result (by field-name selection). Complex records are contructed by two operations—a product (called combination) and a coproduct (called alternation)—and are objects behaving like function products and sums, respectively. Besides its intriguing back-to-front view of functions, an essential difference between Dami's calculus and ours

[3] In other words, the number of slashes is the reference's de Bruijn index.

[4] In a sense, plugging these expressions into the free "outlets!"

is that his calculus does not distinguish variable names and position names—the two are confused. Although quite original and interesting, his calculus departs, unlike ours, in some rather basic way from λ-calculus.[5]

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 we introduce our language of selective λ-terms, and define various selective λ-calculi based on them. In Section 3 we define the selective λ-calculus, including all of them. We add some variations to its syntax, and present some reflections about their meaning. The core of the paper lies in Section 4 where we give the proof of confluence of selective λ-calculus. Finally, we close the paper with some conclusion and a brief discussion of further work to follow this idea in Section 5.

2 Introducing selective λ-calculi

2.1 Generic 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* $\mathcal{L}$.

We will denote variables by x, y, labels in $\mathcal{L}$ by p, q, 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.

2.2 Relative and absolute positions

Before we look at different selective λ-calculi, let us give some intuition to justify this syntax, thinking of two possible sets of labels, symbolic and numeric ones.

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 [19]. 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.[6]

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; *eg.* , $f(a_1, \ldots, a_n)$ may be seen as the more explicit $f(1 \Rightarrow a_1, \ldots, n \Rightarrow a_n)$. 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

[5] Quoting Dami's own words: *"an abstraction (...) is somewhat similar to a λ-abstraction (...). However, there are perhaps more differences than similarities."* Indeed!

[6] 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 [3].

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 $\big(\lambda(1 \Rightarrow x, 2 \Rightarrow y, 4 \Rightarrow z).M\big) \frown (1 \Rightarrow a, 4 \Rightarrow b)$ rather than $(\lambda_1 x.\lambda_1 y.\lambda_2 z.M) \;\widehat{1}\; a \;\widehat{3}\; b$. 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 numeric label the number of numeric-labeled components, smaller than it, and appearing to its left in the labeled Cartesian product. Namely,

$$M \frown (n_1 \Rightarrow N_1, \ldots, n_k \Rightarrow N_k) = M \;\widehat{n'_1}\; N_1 \ldots \;\widehat{n'_k}\; N_k$$
$$\text{where } n'_k = n_k - |\{i \mid i < k, n_i < n_k\}|.$$

Conversely, one may go back from relative syntax to the absolute one by inserting iteratively entities in an abstraction or application tuple. That is,

$$(M \;\widehat{m}\; N) \frown (n_1 \Rightarrow N_1, \ldots, n_k \Rightarrow N_k) = M \frown (m \Rightarrow N, n'_1 \Rightarrow N_1, \ldots, n'_k \Rightarrow N_k)$$
$$\text{where } n'_i = \begin{cases} n_i & \text{if } n_i < m \\ n_i + 1 & \text{if } n_i \geq m \end{cases}$$

These two rules apply directly for abstractions too, and one may verify that they just do opposite work.

For the absolute and relative notations to be effectively coherent, we will expect $M \frown (n_1 \Rightarrow N_1, \ldots, n_k \Rightarrow N_k)$ and $M \frown (n_{\sigma(1)} \Rightarrow N_{\sigma(1)}, \ldots, n_{\sigma(k)} \Rightarrow N_{\sigma(k)})$ to be convertible terms for any permutation σ of $\mathbb{N}_k$, that is, the order of the pairs in a record should be semantically irrelevant.

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.[7] This, however, would be possible only if the order on symbols were not dense. Since, in practice, symbols are 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.

Conversely, one could think of getting rid of numeric labels. However, simply forgetting about numeric labels, just because they are a little cumbersome, would reduce the generality of the calculus. With only symbolic labels we can directly send values to abstractions *as long as they have different labels*. An abstraction can still be hidden by another abstraction with same label. However, with symbolic labels, we have this property *in all cases*. This is certainly useful if you want, for instance, to construct a model of this calculus: intuitively all curried functions become flat, while they would still be partly hierarchized in an keyword-only calculus.

2.3 A lambda-calculus with multiple channels

This is the first possibility, using keywords as label. We define an extension of the lambda calculus, the *symbolic selective λ-calculus*, with symbolic labels.

Following the above syntax, we take our labels from a totally ordered set of symbols $\mathcal{S}$. We will denote these labels by a, b.

To keep compatibility with the classical λ-calculus, we have a default label, 1, such that an unlabeled abstraction or application is interpreted as being labeled by 1.

The reduction rules for this calculus are given in Figure 1. β-reduction only happens on abstractor-applicator pairs with the same label. Otherwise they commute by rule (3). Rules (1) and (2) normalize the order of abstractors and applicators. The condition $x \notin FV(N)$ in rule (3) can always be satisfied through α-conversion.

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

$\beta-$ reduction

(β) $(\lambda_a x.M)\ \widehat{a}\ N \to [N/x]M$

Reordering

(1) $\lambda_a x.\lambda_b y.M \to \lambda_b y.\lambda_a x.M \qquad a > b$

(2) $M\ \widehat{a}\ N_1\ \widehat{b}\ N_2 \to M\ \widehat{b}\ N_2\ \widehat{a}\ N_1 \quad a > b$

(3) $(\lambda_a x.M)\ \widehat{b}\ N \to \lambda_a x.(M\ \widehat{b}\ N) \quad a \neq b, x \notin FV(N)$

Figure 1: Reduction rules for symbolic selective λ-calculus

Definition 1 *We call* symbolic selective λ-calculus *the free combination of rules in Figure 1.*

This calculus is meaningful, in that it is confluent.

Corollary 1 *The symbolic selective λ-calculus is confluent.*

PROOF Consequence of the proof for selective λ-calculus. □

Example 2.1 *We suppose that $a < b < c < d$,*
(For keywords the notations $\lambda(a \Rightarrow x, \ldots)$ and $M(a \Rightarrow N, \ldots)$ are only shorthands.)

$$
\begin{array}{ll}
 & (\lambda(a \Rightarrow x, b \Rightarrow y, c \Rightarrow z).M)\ \widehat{}\ (c \Rightarrow N_1, d \Rightarrow N_2, a \Rightarrow N_3) \\
= & (\lambda_a x.\lambda_b y.\lambda_c z.M)\ \widehat{c}\ N_1\ \widehat{d}\ N_2\ \widehat{a}\ N_3 \\
\to_3 & (\lambda_a x.((\lambda_b y.\lambda_c z.M)\ \widehat{c}\ N_1))\ \widehat{d}\ N_2\ \widehat{a}\ N_3 \\
\to_2 & (\lambda_a x.((\lambda_b y.\lambda_c z.M)\ \widehat{c}\ N_1))\ \widehat{a}\ N_3\ \widehat{d}\ N_2 \\
\to_\beta & (\lambda_b y.\lambda_c z.[N_3/x]M)\ \widehat{c}\ N_1\ \widehat{d}\ N_2 \\
\to_3 & (\lambda_b y.((\lambda_c z.[N_3/x]M)\ \widehat{c}\ N_1))\ \widehat{d}\ N_2 \\
\to_\beta & (\lambda_b y.([N_1/z][N_3/x]M))\ \widehat{d}\ N_2) \\
\to_3 & \lambda_b y.([N_1/z][N_3/x]M\ \widehat{d}\ N_2) \\
= & (\lambda(b \Rightarrow y).(([N_1/z][N_3/x]M)\ \widehat{}\ (d \Rightarrow N_2))
\end{array}
$$

2.4 A lambda-calculus with moving indices

In this calculus we can selectively apply a function on any of its arguments, according to its *apparent position*.

For an unlabeled expression, the apparent position of an abstractor is intuitively defined as the number of times we have to apply this expression in order to have the abstractor applied to the desired argument. For instance, in $\lambda x.\lambda y.\lambda z.M$, the apparent position of the abstractor of z is 3, but in $\lambda x.(\lambda y.\lambda z.M)N$ it is 2. As a consequence, apparent positions do not change when we reduce an expression. When we add labels, we want to keep this property.

Definition 2 Numerical selective λ-calculus *takes its labels from $\mathcal{N} = \mathbb{N} - \{0\}$. Reduction rules on terms modulo α-conversion are given in Figure 2.*

Definition 3 *The* apparent position *of an abstractor in a term M is n such that $M\ \widehat{n}\ N$* associates *this abstractor and N (makes them to be β-reduced together eventually).*

Well-definedness of apparent positions is guaranteed by confluence. Of course, if an abstractor is already linked with an applicator in the term, or appears in the right-hand of an application, it has no apparent position.

Corollary 2 *The numerical selective λ-calculus is confluent.*

$\beta-$ reduction

(β) $(\lambda_n x.M)\ \hat{n}\ N \to [N/x]M$

Reordering

(1) $\lambda_m x.\lambda_n y.M \to \lambda_n y.\lambda_{m-1} x.M \qquad m > n$

(2) $M\ \hat{m}\ N_1\ \hat{n}\ N_2 \to M\ \hat{n}\ N_2\ \widehat{m-1}\ N_1 \quad m > n$

(3) $(\lambda_m x.M)\ \hat{n}\ N \to \lambda_{m-1} x.(M\ \hat{n}\ N) \quad m > n, x \notin FV(N)$

(4) $(\lambda_m x.M)\ \hat{n}\ N \to \lambda_m x.(M\ \widehat{n-1}\ N) \quad m < n, x \notin FV(N)$

Figure 2: Reduction rules for numerical selective λ-calculus

PROOF Consequence of the proof for selective λ-calculus. □

We can now relate apparent positions to the absolute positions in our relative/absolute position dichotomy. The idea is that when we apply M to the tuple $(n_1 \Rightarrow N_1, \ldots, n_k \Rightarrow N_k)$, the n_i's, which are absolute positions in the above definition, are the apparent positions in M of the abstractors they aim at. Similarly, in $\lambda(n_1 \Rightarrow x_1, \ldots, n_k \Rightarrow x_k).M$, x_i has apparent position n_i. As a result, we have

$$(\lambda(n_1 \Rightarrow x_1, \ldots, n_k \Rightarrow x_k).M) \frown (n_1 \Rightarrow N_1, \ldots, n_k \Rightarrow N_k) \xrightarrow{*} [N_i/x_i]_{i=1}^{k} M$$

and the order of bindings in records is free, as one would expect.

Example 2.2 *Numerical indices*

$$\begin{array}{ll}
 & (\lambda(2 \Rightarrow x, 1 \Rightarrow y, 4 \Rightarrow z).M) \frown (4 \Rightarrow N_1, 6 \Rightarrow N_2, 2 \Rightarrow N_3) \\
= & (\lambda_2 x.\lambda_1 y.\lambda_2 z.M)\ \hat{4}\ N_1\ \hat{5}\ N_2\ \hat{2}\ N_3 \\
\to_4 & (\lambda_1 y.\lambda_1 x.\lambda_2 z.M)\ \hat{4}\ N_1\ \hat{5}\ N_2\ \hat{2}\ N_3 \\
\to_7 & (\lambda_1 y.((\lambda_1 x.\lambda_2 z.M)\ \hat{3}\ N_1))\ \hat{5}\ N_2\ \hat{2}\ n_3 \\
\to_5 & (\lambda_1 y.((\lambda_1 x.\lambda_2 z.M)\ \hat{3}\ N_1))\ \hat{2}\ N_3\ \hat{4}\ N_2 \\
\to_7 & (\lambda_1 y.\lambda_1 x.((\lambda_2 z.M)\ \hat{2}\ N_1))\ \hat{2}\ N_3\ \hat{4}\ N_2 \\
\to_\beta & (\lambda_1 y.\lambda_1 x.[N_1/z]M)\ \hat{2}\ N_3\ \hat{4}\ N_2 \\
\to_7 & (\lambda_1 y.((\lambda_1 x.[N_1/z]M)\ \hat{1}\ N_3))\ \hat{4}\ N_2 \\
\to_\beta & (\lambda_1 y.[N_3/x][N_1/z]M)\ \hat{4}\ N_2 \\
\to_7 & \lambda_1 y.([N_3/x][N_1/z]M\ \hat{3}\ N_2) \\
= & \lambda(1 \Rightarrow y).([N_3/x][N_1/z]M \frown (3 \Rightarrow N_2))
\end{array}$$

2.5 A first way to combine these two systems

Intuitively it would be interesting to get in one calculus both the power of symbolic and numeric selective λ-calculi.

For this we take $\mathcal{L}$ to be the disjoint union of the two sets $\mathcal{N}$ of numeric, and $\mathcal{S}$ of symbolic labels. 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.

Our set of rule is the union of numeric (Fig. 2) and symbolic (Fig. 1) rules applied to their respective sets of labels, and the three following rules, which just generalize symbolic ones to handle conflicts with numeric labels, in conformity with our new order.

$(1')$ $\lambda_a x.\lambda_n y.M \to \lambda_n y.\lambda_a x.M$

$(2')$ $M\ \hat{a}\ N_1\ \hat{n}\ N_2 \to M\ \hat{n}\ N_2\ \hat{a}\ N_1$

$(3')$ $(\lambda_a x.M)\ \hat{n}\ N \to \lambda_a x.(M\ \hat{n}\ N) \quad x \notin FV(N)$

We call this system *flat selective λ-calculus*. Again it is confluent.

Corollary 3 *The flat selective λ-calculus is confluent.*

PROOF Consequence of the proof for selective λ-calculus. □

We considered for a long time the flat calculus as the best balanced of these calculi, since it includes both symbolic and numeric calculi in a coherent way. Indeed most of the terms one would write can be expressed in it. However it appears that a stronger one includes it conservatively, and we will rather chose that one as "fundamental" calculus.

3 The selective λ-calculus

3.1 Definition

3.1.1 Syntax

Selective λ-calculus combines orthogonally symbolic and numerical selective λ-calculi. Its set of labels is $\mathcal{L} = \mathcal{S} \times \mathcal{N}$. [8] The order induced on labels is the lexicographical one: $a <_{\mathcal{S}} b \Rightarrow am <_{\mathcal{L}} bn$ and $m <_{\mathcal{N}} n \Rightarrow am <_{\mathcal{L}} an$.

$$M ::= x \mid \lambda_{an} x.M \mid M \ \widehat{an} \ M'$$

The set of selective λ-terms (considered modulo α-conversion) is Λ. We use a, b for symbols (or *channels*), m, n for numbers (or *indices*), and p, q for labels formed of a couple (channel,index).

3.1.2 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 $\mathrm{FV}(M)$ be the set of free variables in M, defined as usual in lambda calculus. The expression $[N/x]M$ denotes the term obtained by replacing all the free occurrences of a variable x by N in (an appropriate α-renaming of) M. That is,

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

3.1.3 Reductions

The reduction system is the combination in Figure 3. We call *weak reordering* the system excepting β-reduction. It may look complex, but one may see reordering rules as structural equalities, and then we have β-reduction as unique reduction rule. One might wonder about why then we do not adopt this view, and separate completely reordering from β-reduction. The answer is that to define from the beginning reordering as a structural equivalence, we would need a good understanding of what is a term modulo reordering. As a matter of fact, we can define it, using a monoid of records. But it would be as complex, and term structure harder to

[8] In [2] this was defined as a product system, and selective λ-calculus as the sum system $\mathcal{L} = \mathcal{S} \cup \mathcal{N}$. Properties of the two systems being similar, we work here on the most general one.

$\beta-$ reduction

(β) $(\lambda_p x.M)\ \widehat{p}\ N \to [N/x]M$

Symbolic reordering

(1) $\lambda_{am}x.\lambda_{bn}y.M \to \lambda_{bn}y.\lambda_{am}x.M$ $\quad a > b$

(2) $M\ \widehat{am}\ N_1\ \widehat{bn}\ N_2 \to M\ \widehat{bn}\ N_2\ \widehat{am}\ N_1$ $\quad a > b$

(3) $(\lambda_{am}x.M)\ \widehat{bn}\ N \to \lambda_{am}x.(M\ \widehat{bn}\ N)$ $\quad a \neq b, x \notin FV(N)$

Numeric reordering

(4) $\lambda_{am}x.\lambda_{an}y.M \to \lambda_{an}y.\lambda_{am-1}x.M$ $\quad m > n$

(5) $M\ \widehat{am}\ N_1\ \widehat{an}\ N_2 \to M\ \widehat{an}\ N_2\ \widehat{am-1}\ N_1$ $\quad m > n$

(6) $(\lambda_{am}x.M)\ \widehat{an}\ N \to \lambda_{am-1}x.(M\ \widehat{an}\ N)$ $\quad m > n, x \notin FV(N)$

(7) $(\lambda_{am}x.M)\ \widehat{an}\ N \to \lambda_{am}x.(M\ \widehat{an-1}\ N)$ $\quad m < n, x \notin FV(N)$

Figure 3: Reduction rules for selective λ-calculus

grasp. So, it is probably more convincing to first verify the confluence on a simple structure, and then trivially extend it to the equational one.

Since the combination is orthogonal (symbolic and numeric labels work on two independent levels), confluence is inherited from the two previous systems.

Theorem 1 *The selective λ-calculus is confluent.*

PROOF in Section 4. □

To let this system include the symbolic and numerical sub-calculi, we will identify a symbol a with the label $(a, 1)$ and an index n with the label $(1, n)$ (this 1 being the default channel of the symbolic calculus). Remark that the default label 1, used to interpret the classical λ-calculus, will result into (1,1) by both rules.

3.2 Entity syntax

To emphasize the similarity between abstraction and application we define a new notation for the first.

$$\lambda_p x.M = M\ \overset{\vee}{p}\ x$$

for any $p \in \mathcal{L}$, $x \in \mathcal{V}$, $M \in \Lambda$.

As a result, β-reduction becomes:

$$M\ \overset{\vee}{p}\ x\ \hat{p}\ N \to_\beta M[x \backslash N],$$

where it is natural to write substitutions on the right side of terms.

We can redefine more clearly the notions we introduced for the generic syntax, and introduce some new ones.

Definition 4 *An* entity *(in Γ) is either an* applicator *or* an abstractor*; that is, a pair consisting of an* operator *($\hat{p}$ or $\overset{\vee}{p}$ for some $p \in \mathcal{L}$) and a term (for applications), or a variable (for abstractions).*

We chose here the term abstractor rather than the usual *binder* to emphasize on the presence of a label. Then we can call binder the variable of an abstractor, that is the x in $\overset{\vee}{p}\ x$.

Definition 5 *We distinguish in a term its* head *and its* spine*. Any selective λ-term can be written*

$$x \odot P = (\dots(x\ e_1)\dots)e_n$$

where the head x is a variable and the spine P is an entity sequence $(e_1, \dots, e_n) \in \Gamma^*$.

We shall write $\iota_P(e_k) = n - k$ to denote the position *of entity e_n in spine P (counting from the right).*

- For any entity sequence P in Γ^* we define $BV(P)$, the set of bound variables in P; that is, the set of variables abstracted by entities forming P (or, all the variables that are immediately preceded by a $\overset{\vee}{p}$ operator in P). For any M in Λ, any occurrence of a variable of $BV(P)$ in M is bound in $M \odot P$.
- We note $P \cdot Q$ the concatenation of two entity sequences, and have $M \odot (P \cdot Q) = (M \odot P) \odot Q$.

Example 3.1 *In the term* $x \hat{p} z \overset{\vee}{q} x \overset{\vee}{q} y$*,* x *is the* head *and* $P = \hat{p} y \overset{\vee}{q} x \overset{\vee}{q} y$ *the spine.* $BV(P)$ *is* $\{x, y\}$ *and* $FV(P)$ *is* $\{z\}$*.*

This notation will simplify many proofs. We will use indifferently the two notations in the following.

3.3 Towards a transformation calculus

Before detailing the proof of confluence, we will sketch some interesting extensions of selective λ-calculus. We will start from the new notation introduced above, and explain in what way this notation suggests another extension.

The intuition behind selective λ-calculus is no longer functions but functions over labeled arguments which behave like communicating processes through named and (relatively) indexed 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 $f \circ g = \lambda x.f(gx)$ in the classical calculus. But in the selective λ-calculus we would have to parameterize this composition with three labels! That is $f \circ_{pqr} g = f \hat{p} (g \hat{q} x) \overset{\vee}{r} x$, the function taking its input on r, giving it to g on q, and feeding the result to f on p. This is complex, and this is rather weak. Particularly when we think of the powerful out-of-order currying capabilities of our system.

Rather, we will just use syntactic juxtapostion $\odot$, as defined above. Then we can obtain a more interesting composition with $M \circ P = M \odot P$, 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. This lets us define a *transformation calculus* [11], able to describe notions like state in a lambda calculus framework. More immediately, 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 [19], although still a direct and conservative extension of classical λ-calculus, and particularly still confluent, thanks to our indices. 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.

4 Proof of confluence

4.1 Pseudo-reduction

Reordering rules may be viewed as defining a syntactic congruence on terms [19]. Reduction must then be shown to be confluent modulo this congruence.

4.1.1 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,

$$(a) \quad M \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N_1 \overset{\wedge}{}_q\, N_2 \to M \overset{\wedge}{}_q\, N_2 \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N_1 \quad x \notin FV(N_2)$$
$$(b) \quad M \overset{\vee}{}_q\, y \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N \to M \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N \overset{\vee}{}_q\, y \quad y \notin FV(N)$$

Example 4.1 *If we use only reordering rules,* $(\lambda_1 x.\lambda_2 y.x \overset{\wedge}{}_1\, y) \overset{\wedge}{}_2\, a \overset{\wedge}{}_1\, b$ *can be reduced by Rules* (7) *and* (6) *yielding* $A = (\lambda_1 x.\lambda_1 y.x \overset{\wedge}{}_1\, y \overset{\wedge}{}_1\, a) \overset{\wedge}{}_1\, b$*. It can also be reduced by Rule* (5) *to* $B = (\lambda_1 x.\lambda_2 y.x \overset{\wedge}{}_1\, y) \overset{\wedge}{}_1\, b \overset{\wedge}{}_1\, a$*. 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*.*

4.1.2 Pseudo-reduced form equivalence (PRF)

Since some critical pairs appear between reordering rules, we introduce an equivalence relation that unifies their results. Confluence of the reordering part of the system can only be considered under this equivalence.

Definition 6 *If* $x \notin FV(N_1)$ *and* $y \notin FV(N_2)$ *then*

$$M \overset{\vee}{}_q\, y \overset{\wedge}{}_q\, N_1 \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N_2 \leftrightarrow M \overset{\vee}{}_p\, x \overset{\wedge}{}_p\, N_2 \overset{\vee}{}_q\, y \overset{\wedge}{}_q\, N_1$$

For any M, x, N*,*

$$M \overset{\vee}{}_{am}\, x \overset{\wedge}{}_{am}\, N \leftrightarrow M \overset{\vee}{}_{an}\, x \overset{\wedge}{}_{an}\, N$$

Example 4.2 *Consider the term* $A = M \overset{\vee}{}_1\, x \overset{\vee}{}_2\, y \overset{\wedge}{}_2\, N \overset{\wedge}{}_1\, N'$*. If we exclude* β*-reduction, we can still apply either rule (4) or (5), yielding to* $B = M \overset{\vee}{}_1\, y \overset{\vee}{}_1\, x \overset{\wedge}{}_2\, N \overset{\wedge}{}_1\, N'$ *or* $C = M \overset{\vee}{}_1\, x \overset{\vee}{}_2\, y \overset{\wedge}{}_1\, N' \overset{\wedge}{}_1\, N$*, which by (7) and (6) give* $B' = M \overset{\vee}{}_1\, y \overset{\wedge}{}_1\, N \overset{\vee}{}_1\, x \overset{\wedge}{}_1\, N'$ *and* $C' = M \overset{\vee}{}_1\, x \overset{\wedge}{}_1\, N' \overset{\vee}{}_1\, y \overset{\wedge}{}_1\, N$*. No reordering rule can recover confluence between* B' *and* C'*, but they are PRF equivalent.*

The necessity for the second equation is more subtle, but is linked to the use of pseudo-reductions.

Combining these two we can see that PRF equivalence makes both order and symbolic part of labels irrelevant for *locally associated pairs* (*cf.* Def. 8), as long as variable dependencies are satisfied.

4.2 Combined systems and restricted reductions

We will first distinguish reordering rules, and prove properties of those alone. 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 weak reordering combined with pseudo-reduction. We will call it the *reordering system*.

An *order normal form* is a normal form for the reordering system.

A *stable reordering* is a reordering where pseudo-reduction is prefered to weak reordering for critical pairs.

Intermediate statements will be done on the system called β*-reordering*. It is β-reduction on order normal forms, where the result of each step is normalized by the reordering system.

A last system, the *label-parallel system*, which makes the link between selective λ-calculus and β-reordering, is the combination of all rules, and the *stable label-parallel system* includes the same restriction as stable reordering.

For each of these reduction systems, we shall use the symbol $\to$ to indicate a single reduction step using any of system's rules, and $\to_r$ if the rule uses Rule (r). When unconcerned by termination, we shall accept the step $(M \to M)$ in this relation. As usual, $\overset{*}{\to}$ 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 $\overset{*}{\to}$ using only ϱ-reduction steps. For example, $\triangleright_{stb}$ for stable reorderings, $\triangleright_{mcd}$ for minimal complete developments, *etc.*...

For many of these systems confluence will be considered *modulo* pseudo-reduced form equivalence. Here is the definition, as given in [15].

Definition 7 *We say that the relation* $\to$ *is* confluent modulo $\sim$ *iff*

$$(\forall xyx'y')\ x \sim y \wedge x \overset{*}{\to} x' \wedge y \overset{*}{\to} y' \Rightarrow (\exists \overline{xy})\ x' \overset{*}{\to} \overline{x} \wedge y' \overset{*}{\to} \overline{y} \wedge \overline{x} \sim \overline{y}.$$

4.3 Confluence of the reordering system

Before going on it may be good to have an idea of what an order normal form looks like, but first we need a basic definition.

Definition 8 *An abstractor and an applicator are said to be* locally associated *when they match β-reduction, namely $\overset{\vee}{p}\ x$ and $\hat{p}\ N$ in $M\ \overset{\vee}{p}\ x\ \hat{p}\ N$. They are* locally free *if they do not. They are* associated *if a reordering can bring them to this state,* free *otherwise.*

Proposition 1 *The general structure of an order normal form is:*

$$x\ \overset{\wedge}{p_1}\ N_1 \dots \overset{\wedge}{p_j}\ N_j\ \overset{\vee}{q_k}\ y_k\ \overset{\wedge}{q_k}\ K_k \dots \overset{\vee}{q_1}\ y_1\ \overset{\wedge}{q_1}\ K_1\ \overset{\vee}{r_l}\ x_l \dots \overset{\vee}{r_1}\ x_1$$
where $p_i \leq p_{i+1}$, $r_i \leq r_{i+1}$, N_i's and K_i's are in order normal form.

PROOF Thanks to rules (3),(6),(7) and pseudo-reductions, in each spine, all locally free applicators (resp. abstractors) have to be on the left of all absractors (resp. on the right of all applicators).

By rules (1) and (4) free abstractors must have decreasing labels; and by rules (2) and (5) free applicators must have growing ones.

Locally associated abstractors and applicators stay by pair with the same label. □

Then next lemma is essential, since it allows us to consider reordering as a reduction on independent spines rather than terms.

Lemma 1 (Stability of entities) *The reordering rules (1)–(7),$(a), (b)$ 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, and only the numeric part of its label may change.*

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 $n-1$). □

Now we can give a new, and more general definition of the notion of apparent position. We base it on *shifting*.

Definition 9 *To each spine P we associate a* shifting function $\phi(P)$ *defined as follows. It corresponds to pushing $\overset{\vee}{am}\ x$ to the right in P, using reordering rules bidirectionally.*

$$\begin{array}{lcll}
\phi()(am) & = & am & \\
\phi(\overset{\vee}{bk}\ y \cdot P)(am) & = & \phi(P)(am) & a \neq b \\
\phi(\overset{\wedge}{bk}\ N \cdot P)(am) & = & \phi(P)(am) & a \neq b \\
\phi(\overset{\vee}{an}\ y \cdot P)(am) & = & \phi(P)(am) & m < n \\
\phi(\overset{\vee}{an}\ y \cdot P)(am) & = & \phi(P)(am+1) & m \geq n \\
\phi(\overset{\wedge}{an}\ N \cdot P)(am) & = & \phi(P)(am) & m < n \\
\phi(\overset{\wedge}{an}\ N \cdot P)(am) & = & \phi(P)(am-1) & m > n
\end{array}$$

Note that $\phi(P)(am)$ is undefined iff our abstractor encounters an applicator with same label in P. Similarly $\phi^{-1}(P)(am)$ is uniquely defined and corresponds to pushing $\overset{\wedge}{am}\ N$ to the left through P; it is undefined iff our applicator encounters an abstractor with same label.

The *dual* $\overline{M}$ of a term M is defined by:

$$\begin{array}{lcl}
\overline{P \cdot Q} & = & \overline{Q} \cdot \overline{P} \\
\overline{\overset{\vee}{an}\ x} & = & \overset{\wedge}{an}\ x \\
\overline{\overset{\wedge}{an}\ N} & = & \overset{\vee}{an}\ x_N
\end{array}$$

Proposition 2 (shifting)

a. ϕ *is compositional:* $\phi(P \cdot Q) = \phi(Q) \circ \phi(P)$.

b. $\phi^{-1}(P) = \phi(\overline{P})$ *where* $\overline{P}$ *is the dual of* P.

PROOF

a. by trivial induction.

b. by verifying cases 5 and 7 of the definition exchange correctly, and by compositionality of ϕ.

□

Definition 10 *The apparent position of* $\overset{\vee}{_{an}}\ x$ *in* $P = P_1 \cdot \overset{\vee}{_{an}}\ x \cdot P_2$ *is* $\pi_P(\overset{\vee}{_{an}}\ x) = \phi(P_2)(an)$. *That of* $\overset{\wedge}{_{an}}\ N$ *is* $P = P_1 \cdot \overset{\wedge}{_{an}}\ N \cdot P_2$ *is* $\pi_P(\overset{\wedge}{_{an}}\ N) = \phi^{-1}(P_1)(an)$.

We will have to verify that this new definition matches the precedent. This is in fact equivalent to having the following *static association* match previous association.

Definition 11 *An abstractor* $\overset{\vee}{_{am}}\ x$ *and an applicator* $\overset{\wedge}{_{an}}\ N$ *are said to be* statically associated *when they belong to the same spine* $P \cdot \overset{\vee}{_{am}}\ x \cdot Q \cdot \overset{\wedge}{_{an}}\ N \cdot R$ *and* $\phi(Q)(an) = am$.

What intuitionally this definition does is using ϕ to simulate a reordering moving $\overset{\vee}{_{am}}\ x$ outwards. If this simulation succeeds in meeting $\overset{\wedge}{_{an}}\ N$ with an abstractor $\overset{\vee}{_{an}}\ x$, then they are statically associated.

The following lemma is the most important of this subsection. It proves all at once that apparent positions are invariant, static association too, and that it is association.

Lemma 2 *Apparent positions are conserved by reordering.*

PROOF We only study the case for an abstractor, since we can extend to applicators by duality (Proposition 2b, all reordering rules being symmetrical w.r.t. abstractors and applicators).

In $P \cdot \overset{\vee}{_{am}}\ x \cdot Q$, we have three possible positions of the reordering redex:

1. either the redex is included in P, and has no effect on the apparent position,
2. either it is included in $Q = Q_1 \cdot R \cdot Q_2$, with R the redex, and we need to prove that $\phi(R') = \phi(R)$, since $\phi(Q) = \phi(Q_2) \circ \phi(R) \circ \phi(Q_1)$,
3. either it contains $\overset{\vee}{_{am}}\ x$.

We first prove that for any reordering redex R, $\phi(R')(ck) = \phi(R)(ck)$. We proceed by case on the rules:

- Symbolic rules. We only detail the first one: $\overset{\vee}{_{bn}}\ y\ \overset{\vee}{_{am}}\ x \to \overset{\vee}{_{am}}\ x\ \overset{\vee}{_{bn}}\ y$. If $c \neq a$ and $c \neq b$ then $\phi(R')(ck) = \phi(R)(ck) = ck$. If $c = a$ then $\phi(R')(ak) = \phi(R)(ak) = \phi(\overset{\vee}{_{am}}\ x)(ak)$. Resp. for $c = b$.

  Others use the same argument (one interference in maximum, and invariant).

- Numeric rules. If $c \neq a$ then there is no interference: $\phi(R')(ck) = \phi(R)(ck) = ck$. Otherwise we

calculate the interference case by case.

(4)	$\phi({}^{\vee}_{an}\ y\ {}^{\vee}_{am}\ x)(ak)$			$\phi({}^{\vee}_{am-1}\ x\ {}^{\vee}_{an}\ y)(ak)$
$k \geq m-1 \geq n$	$\phi({}^{\vee}_{am}\ x)(ak+1)$	$=$	$ak+2$	$=$ $\phi({}^{\vee}_{an}\ y)(ak+1)$
$m-1 > k \geq n$	$\phi({}^{\vee}_{am}\ x)(ak+1)$	$=$	$ak+1$	$=$ $\phi({}^{\vee}_{an}\ y)(ak)$
$m-1 \geq n > k$	$\phi({}^{\vee}_{am}\ x)(ak)$	$=$	ak	$=$ $\phi({}^{\vee}_{an}\ y)(ak)$
(5)	$\phi({}^{\wedge}_{am}\ N_1\ {}^{\wedge}_{an}\ N_2)(ak)$			$\phi({}^{\wedge}_{an}\ N_2\ {}^{\wedge}_{am-1}\ N_1)(ak)$
$k > m > n$	$\phi({}^{\wedge}_{an}\ N_2)(ak-1)$	$=$	$ak-2$	$=$ $\phi({}^{\wedge}_{am-1}\ N_1)(ak-1)$
$m > k > n$	$\phi({}^{\wedge}_{an}\ N_2)(ak)$	$=$	$ak-1$	$=$ $\phi({}^{\wedge}_{am-1}\ N_1)(ak-1)$
$m > n > k$	$\phi({}^{\wedge}_{an}\ N_2)(ak)$	$=$	ak	$=$ $\phi({}^{\wedge}_{am-1}\ N_1)(ak)$
(6)	$\phi({}^{\vee}_{am}\ x\ {}^{\wedge}_{an}\ N)(ak)$			$\phi({}^{\wedge}_{an}\ N\ {}^{\vee}_{am-1}\ x)(ak)$
$k \geq m > n$	$\phi({}^{\wedge}_{an}\ N)(ak+1)$	$=$	ak	$=$ $\phi({}^{\vee}_{am-1}\ x)(ak-1)$
$m > k > n$	$\phi({}^{\wedge}_{an}\ N)(ak)$	$=$	$ak-1$	$=$ $\phi({}^{\vee}_{am-1}\ x)(ak-1)$
$m > n > k$	$\phi({}^{\wedge}_{an}\ N)(ak)$	$=$	ak	$=$ $\phi({}^{\vee}_{m-1}\ x)(ak)$
(7)	$\phi({}^{\vee}_{am}\ x\ {}^{\wedge}_{an}\ N)(ak)$			$\phi({}^{\wedge}_{an-1}\ N\ {}^{\vee}_{am}\ x)(ak)$
$k > n-1 \geq m$	$\phi({}^{\wedge}_{an}\ N)(ak+1)$	$=$	ak	$=$ $\phi({}^{\vee}_{am}\ x)(ak-1)$
$n-1 > k \geq m$	$\phi({}^{\wedge}_{an}\ N)(ak+1)$	$=$	$ak+1$	$=$ $\phi({}^{\vee}_{am-1}\ x)(ak)$
$n > m > k$	$\phi({}^{\wedge}_{an}\ N)(ak)$	$=$	ak	$=$ $\phi({}^{\vee}_{m-1}\ x)(ak)$

- Pseudo-reductions. For any pair $P = {}^{\vee}_{an}\ x\ {}^{\wedge}_{an}\ N$, $\phi(P) = Id$: this is clear for a label with a different symbolic part. Otherwise, either $m < n$, and $\phi(P)(am) = \phi({}^{\wedge}_{an}\ N)(am) = am$, or $m \geq n$, and $\phi(P)(am) = \phi({}^{\wedge}_{an}\ N)(am+1) = am$.

  As a result, since pseudo-reductions do not change indices, $\phi(R') = \phi(R) = \phi({}^{\wedge}_{q}\ N_2)$ (resp. $\phi({}^{\vee}_{q}\ y)$).

When the reordering redex contains ${}^{\vee}_{am}\ x$, we verify that they stay associated. Pseudo-reductions and symbolic reordering are clearly not a problem, since the former moves a locally assocated pair, with no effect on ϕ, and the later move an entity with a different symbolic part.

For the numeric rules we just remark that ϕ simulates a reduction moving ${}^{\vee}_{am}\ x$ outwards, using the rules bidirectionally. As such all steps are either the actual simulated step (outward) or its opposite (inward), and naturally static association is conserved. □

Corollary 4 *Static associations are conserved by reordering. Static association is association.*

PROOF The spine is $P\cdot\ {}^{\vee}_{am}\ x \cdot Q\cdot\ {}^{\wedge}_{an}\ N \cdot R$.

When the reordering redex contains both ${}^{\vee}_{am}\ x$ and ${}^{\wedge}_{an}\ N$, this is a pseudo-reduction, and they stay locally associated, which is statically associated.

When the reordering redex is contained in $P\cdot\ {}^{\vee}_{am}\ x \cdot Q$, we apply Lemma 2 on it.

Otherwise we apply the lemma on $Q\cdot\ {}^{\wedge}_{an}\ N \cdot R$.

Moreover, by Proposition 1, in an order normal form all entities are either free or locally associated. So, statically associated entities associated.

Reciprocally, since statically free entities have an apparent position out of the spine, they cannot be associated. As such all associated entities are statically associated. □

With these lemma and corollary we easily prove the confluence of reordering modulo PRF equivalence, once we have termination.

Proposition 3 *The reordering system in Noetherian.*

PROOF We define a mesure on spines by the following ordered pair:[9]

$$\begin{aligned}\mu(P) = \quad & (|\{(e,e') \in P \mid e \text{ abstractor}, e' \text{ applicator}, \iota_P(e) > \iota_P(e')\}|, \\ & |\{(e,e') \in P \mid e,e' \text{ abstractors}, \iota_P(e) > \iota_P(e'), \pi_{P[e,e']}(e) < \pi_{P[e,e']}(e')\}| \\ & +|\{(e,e') \in P \mid e,e' \text{ applicators}, \iota_P(e) > \iota_P(e'), \pi_{P[e,e']}(e) > \pi_{P[e,e']}(e')\}|)\end{aligned}$$

where $P[e,e']$ is the sub-spine extracted from P between e an e' (included). When one of the π's is not defined, the inequality is considered false.

Now we must prove that this mesure, as lexicographical ordering, decreases.

For the first term this is easy: only rules (3),(6),(7) and pseudo-reductions may change it, and they reduce it.

For the second one we must be more careful, because of the changing sub-spine in π. But we remark that $\phi(P)$ is strictly monotonous for any P, and by compositionality we can extend our sub-spine to the whole redex and end up with the same order. However, a special case arises when this is a pseudo-reduction with e or e' locally associated. We can no longer use ϕ. Actually, this may increase the term. However, since the first term increased, this does not matter.

Moreover, as needed, rules (1),(4) and (2),(5) decrease respectively the left and right side of the sum, while clearly not changing the other.

Since all rules decrease that mesure, which is well-founded, reordering terminates. □

Theorem 2 *The reordering system is confluent modulo PRF equivalence.*

PROOF We prove this property spine by spine, since reordering keeps entities on the same spine (*cf.* Lemma 1).

Since by Proposition 3 we know that reordering terminates, we just have to prove that for any spine taken modulo PRF equivalence, its order normal form is unique modulo PRF equivalence.

By Lemma 2 and Corrolary 4, we know that both apparent positions for free entities, and associations for associated ones, are invariant by reordering. We verify easily that they are invariant by PRF equivalence too.

Moreover Proposition 1 gives us the structure of order normal forms. First free abstractor and free applicator parts are entirely specified by there apparent positions, and the ordering on labels. Then PRF equivalence says that the order of locally associated pairs is irrelevant, and the numeric part of their labels too. Since we know by Lemma 1 that the symbolic part does not change, that specifies the associated part modulo PRF equivalence. Since all reordering rules respect variable dependencies, they are kept.

As a result order normal forms are completely specified, modulo PRF equivalence, by the original term. □

4.4 Confluence of β-reordering

Definition 12 (β-Reordering) *A* β-reordering *step is a β-reduction step immediately followed by a stable reordering to order normal form.*

Since stable reordering can reduce every time reordering reduces, and 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 12 to stand we have to prove that PRF-equivalent order-normal expressions are still equivalent after β-reordering. This is immediate, since PRF-equivalence and β-reduction are orthogonal: β-reduction do not separate any association pair, and PRF-equivalence do not separate any β-redex.

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 the PRF equivalence class of its order normal forms,

[9] Recall that positions (ι_P) in a spine start from the right.

or $M\downarrow$, implicitly. We shall do that everywhere in the section, except of course when we prove properties about it. That is, everywhere except Lemmas 3, 4 and Proposition 4 (which is unrelated). 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 [14]. By *contracting* a β-redex, we mean applying the corresponding step of β-reduction.

Definition 13 (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 $S \neq R$. Then R has form $(\lambda_p x.M)\ \widehat{p}\ N$ and S is in M or in N. If S is in M, then $S' = [x/N]S$. If S is in N, then there are as many residuals S' of S as there were occurences of x in M, and $S' = S$.*

  This last 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 associated pairs of entities such that R and S are not overlapping. Note also that in the first three cases S has at most one residual.

Before going on with our proof about β-reordering, we enunciate finite developments for β alone.

Proposition 4 (finite developments) *Let R be a set of β-redexes in M. Then reducing, in any order, of all residuals of redexes in R terminates and converges to the same M'.*

PROOF Since we use only β-reduction, finite developments for classical lambda calculus does apply. □

Now we are back to our proof about β-reordering. However, the above definition of residuals does not take into account that we are working with order-normal terms. In fact we will use here a more subtle definition of β-redex.

Definition 14 (β-redex) *In $(\lambda_p x.(y \odot P))\ \widehat{p}$ only are included in the β-redex, y if $y = x$, all applicators in P containing x, and all abstractors in P whose associated applicator is in the β-redex.*

With this definition residuals are well defined: we have just to replace β-reduction by β-reordering.

Lemma 3 (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 heads of spines 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 4 (Induction)

$$M \rightarrow_{\beta\downarrow} M' \Rightarrow \lambda_p x.M \rightarrow_{\beta\downarrow} \lambda_p x.M'$$
$$M \rightarrow_{\beta\downarrow} M' \Rightarrow M\ \widehat{p}\ N \rightarrow_{\beta\downarrow} M'\ \widehat{p}\ N$$
$$N \rightarrow_{\beta\downarrow} N' \Rightarrow M\ \widehat{p}\ N \rightarrow_{\beta\downarrow} M\ \widehat{p}\ N'$$

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 $p_i \leq p_{i+1}$. So that $M' = \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.M_1'$, with M_1' any order-normal expression, and $M_1 \rightarrow_{\beta\downarrow} M_1'$. Hence

$$\begin{array}{lll} \lambda_p x.M & = & \lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1 \\ & \rightarrow_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1' \\ & = & \lambda_p x.M' \end{array}$$

2. If $\widehat{p}\ N$ is associated then

$$\begin{array}{lll} M\ \widehat{p}\ N & = & (\lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1)\ \widehat{p}\ N \quad M_1 \text{ as before}, p_i \leq p_{i+1} \\ & = & \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.((\lambda_{p'} x.M_1)\ \widehat{p'}\ N) \quad \text{(order normal form)} \\ & \rightarrow_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.((\lambda_{p'} x.M_1')\ \widehat{p'}\ N) \\ & = & (\lambda_{p_1} x_1 \ldots \lambda_{p'} x \ldots \lambda_{p_n} x_n.M_1')\ \widehat{p}\ N \quad M_1' \text{ order normal} \\ & = & M'\ \widehat{p}\ N \end{array}$$

   If $\widehat{p}\ N$ is not associated then

$$\begin{array}{lll} M\ \widehat{p}\ N & = & (\lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.A(z\ \widehat{q_1}\ N \ldots\ \widehat{q_m}\ N_m))\ \widehat{p}\ N \\ & = & \lambda_{p_1} x_1 \ldots \lambda_{p_n'} x_n.A(z\ \widehat{q_1}\ N \ldots\ \widehat{p'}\ N\ \widehat{q_m'}\ N_m) \\ & \rightarrow_{\beta\downarrow} & \lambda_{p_1} x_1 \ldots \lambda_{p_n'} x_n.A'(Z'\ \widehat{q_1}\ N_1' \ldots\ \widehat{p'}\ N \ldots\ \widehat{q_m'}\ N_m') \\ & = & (\lambda_{p_1} x_1 \ldots \lambda_{p_n} x_n.A'(Z'\ \widehat{q_1}\ N_1' \ldots\ \widehat{q_m}\ N_m'))\ \widehat{p}\ N \\ & = & M'\ \widehat{p}\ N \end{array}$$

   where A is the entity associations, A' the reduced associations, 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 it properly contains no other R_j (using our new definition of β-redex).

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 $n - 1$ residuals $R_2', \ldots, R_n'$, 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 15 (MCD) *Let P be a term as above, and Q a term. We write $P \rhd_{mcd} Q$ iff Q is obtained from P by* minimal complete development *of the set $\{R_1, \ldots, R_n\}$.*

Note that if $M \rhd_{mcd} M'$ and $N \rhd_{mcd} N'$, then $M\ \widehat{p}\ N \rhd_{mcd} M'\ \widehat{p}\ N'$. (*cf.* , Lemma 4)

Lemma 5 *If $M \rhd_{mcd} M'$ and $N \rhd_{mcd} N'$, then*

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

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

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

2. $x \notin \mathrm{FV}(M)$. Then $x \notin \mathrm{FV}(M')$, so

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

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 $M_1 \rhd_{mcd} M_1'$. Hence

$$\begin{array}{lll} [N/x]M & = & [N/x](\lambda_p y.M_1) \quad \text{Lemma 3} \\ & = & \lambda_p y.[N/x]M_1 \quad \text{since } y \notin \mathrm{FV}(xN) \\ & \rhd_{mcd} & \lambda_p y.[N'/x]M_1' \quad \text{by induction hypothesis} \\ & = & [N'/x]M' \quad \text{since } y \notin \mathrm{FV}(xN') \end{array}$$

4. $M = M_1 \ \hat{p} \ M_2$ and each R_i is in M_1 or M_2. Then M' has form $M_1' \ \hat{p} \ M_2'$ where $M_j \rhd_{mcd} M_j'$ for $j = 1, 2$. Hence

$$\begin{array}{lll} [N/x]M & = & ([N/x]M_1) \ \hat{p} \ ([N/x]M_2) \quad \text{Lemma 3} \\ & \rhd_{mcd} & ([N'/x]M_1') \ \hat{p} \ ([N'/x]M_2') \quad \text{by ind. and note above} \\ & = & [N'/x]M'. \end{array}$$

5. $M = (\lambda_p y.L) \ \hat{p} \ Q$ 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 $M = (\lambda_q z.K) \ \hat{q} \ O$ too, and this one is contracted last). Hence the MCD has form

$$\begin{array}{lll} M = (\lambda_p y.L) \ \hat{p} \ Q & \rhd_{mcd} & (\lambda_p y.L') \ \hat{p} \ Q' \quad (L \rhd_{mcd} L', Q \rhd_{mcd} Q') \\ & \rightarrow_{\beta\downarrow} & [Q'/y]L' \\ & = & M'. \end{array}$$

By induction hypothesis we have MCD's of $[N/x]L$ and $[N/x]Q$. Hence

$$\begin{array}{lll} [N/x]M & = & (\lambda_p y.[N/x]L) \ \hat{p} \ ([N/x]Q) \quad \text{since } y \notin \mathrm{FV}(xN) \\ & \rhd_{mcd} & (\lambda_p y.[N'/x]L') \ \hat{p} \ ([N'/x]Q') \quad \text{induction} \\ & \rightarrow_{\beta\downarrow} & [([N'/x]Q')/y][N'/x]L' \\ & = & [N'/x][Q'/y]L' \\ & = & [N'/x]M'. \end{array}$$

This reduction is an MCD, as required.

□

The following lemma is necessary because we are working on order normal forms: it is unclear wether $(\lambda_p x.M) \ \hat{p} \ N$ will still be most external after reordering, but thanks to our definition of redex, it cannot be included in any other, so can be reduced last.

Lemma 6 (Proof induction) *If there is an MCD*

$$\begin{array}{lllll} P & = & (\lambda_p x.M) \ \hat{p} \ N & \rightarrow\!\!\!\!^{*}_{\beta\downarrow} & (\lambda_p x.M') \ \hat{p} \ N' \\ & & & \rightarrow_{\beta\downarrow} & [N'/x]M' \quad = \ Q \\ & & & \rightarrow\!\!\!\!^{*}_{\beta\downarrow} & Q' \end{array}$$

then there is an MCD

$$\begin{array}{lllll} P & = & (\lambda_p x.M) \ \hat{p} \ N & \rightarrow\!\!\!\!^{*}_{\beta\downarrow} & (\lambda_p x.M'') \ \hat{p} \ N'' \\ & & & \rightarrow_{\beta\downarrow} & [N''/x]M'' \quad = \ Q' \end{array}$$

That is, reduction of a potentially most external redex may be done last.

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 5.

Each step of the original MCD after $[N'/x]M'$ only modifies either N' or M' at a time. So that we can write $M' \to_{\beta\downarrow} M_1 \to_{\beta\downarrow} \ldots \to_{\beta\downarrow} M''$, 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 7 (confluence of MCD) *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 6, there is an MCD with form

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

   And the other MCD has form

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

   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 5

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

   And for B,

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

5. $P = (\lambda_p x.M)\ \widehat{p}\ N$ and both the given MCD's contract P's residual. Then (Lemma 6) we can give these MCD's form

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

Apply the induction hypothesis to M and N in case 4, and choose $T = [N^+/x]M^+$. Then Lemma 5 gives the result, as above.

□

Theorem 3 *β-reordering is confluent modulo PRF equivalence.*

$$P \stackrel{*}{\to}_{\beta\downarrow} M,\ P \stackrel{*}{\to}_{\beta\downarrow} N \Rightarrow (\exists T)\ M \stackrel{*}{\to}_{\beta\downarrow} T,\ N \stackrel{*}{\to}_{\beta\downarrow} T.$$

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

$$P \to_{\beta\downarrow},\ P \stackrel{*}{\to}_{\beta\downarrow} N \Rightarrow (\exists T)\ M \stackrel{*}{\to}_{\beta\downarrow} T,\ N \stackrel{*}{\to}_{\beta\downarrow} T.$$

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

$$P \triangleright_{mcd} M,\ P \stackrel{*}{\to}_{\beta\downarrow} N \Rightarrow (\exists T)\ M \stackrel{*}{\to}_{\beta\downarrow} T, N \triangleright_{mcd} T.$$

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

4.5 Confluence of selective λ-calculus

In this section, $\to$ (or $\to_\lambda$) denotes the union of β-reduction and ordering rules (label-selective λ-calculus), and $\to_\omega$ 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 4.

Definition 16 (Normalized reduction) *For each label-parallel reduction $M_0 \to_\omega M_1 \to_\omega \ldots \to_\omega M_n$ we define its* normalized reduction $N_0 \to_{\beta\downarrow} N \to_{\beta\downarrow} \ldots \to_{\beta\downarrow} N_n$ *by taking for each N_i the order-normal form $M_i\downarrow$.*

Proposition 5 *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 Corollary 4, all β-redexes in M_i are still β-redexes in N_i.

If $M_i \to M_{i+1}$ is a reordering step, then $N_i = N_{i+1}$. Else, $M_i \to M_{i+1}$ is a β-step, and we should show $N_i \to_{\beta\downarrow} N_{i+1}$. From our remark, we have $N_i \to_{\beta\downarrow} N_i'$, reducing the same redex. We will in fact construct two parallel reorderings of M_i and M_{i+1}. First, a stable reordering of M_i, from $M_i^0 = M_i$ to $M_i^k = N_i$. With such a reordering, we have at each step $M_i^j \to M_i'^j$ by a β-step. Then we define a reordering of M_{i+1} going through all $M_i'^j$'s. By definition $M_i^j \to M_i^{j+1}$ does not separate two locally associated entities. There are four cases to consider:

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

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

Theorem 4 (Confluence of label-parallel reduction) *The label-parallel system is confluent. Moreover, the converging reductions are* stable*,*

$$P \xrightarrow{*}_{\omega} M,\ P \xrightarrow{*}_{\omega} N \ \Rightarrow\ (\exists T)\ M \rhd_{\omega stb} T,\ N \rhd_{\omega stb} T.$$

PROOF We have

$$\begin{array}{llll} P & \to_{\omega} M_1 & \to_{\omega} \ldots \to_{\omega} M_m & = M, \\ P & \to_{\omega} N_1 & \to_{\omega} \ldots \to_{\omega} N_n & = N. \end{array}$$

So that we obtain normalized reductions

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

And by confluence of β-reordering modulo PRF-equivalence,

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

with T and T' PRF-equivalent.

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

We can then reduce all the β-redexes present in the resulting term, thanks to Proposition 4, 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 1 (Confluence of label-selective λ-calculus) *The label-selective λ-calculus is confluent. That is,*

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

PROOF By Theorem 4,

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

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.

We first define the set B_k of all residuals of β-redexes which where implied in an (a) or (b) pseudo-reduction. That is $B_0 = \emptyset$, $B_{k+1} = \{\text{residuals of } B_k \text{ in } R_k \to R_{k+1}\}$ if this was not a pseudo-reduction, $B_{k+1} = \{\text{residuals of } B_k \text{ in } R_k \to R_{k+1}\} \cup \{\text{the skipped } \beta\text{-redex}\}$ if it was. Since reductions are stable, residuals do not disappear.

Then we define R'_k as R_k where all redexes in B_k were reduced; Proposition 4 makes this definition correct. We have $R'_k \xrightarrow{*} R'_{k+1}$ where $\to$ is either the original $R_k \to R_{k+1}$ step applied on all its residuals, either $\to_\beta$ applied on $B_{k+1} \setminus B_k$ if it was a pseudo-reduction.

We define similarly S'_k.

We will finally have two expressions, coming from 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, since the B_k's contain only residuals of β-redexes. That is,

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

So that finally, $M \xrightarrow{*} T$ and $N \xrightarrow{*} T$. □

Figure 4 shows a schematic diagram of the process.

Corollary 1,2,3 *Symbolic, numeric and flat selective λ-calculi are confluent.*

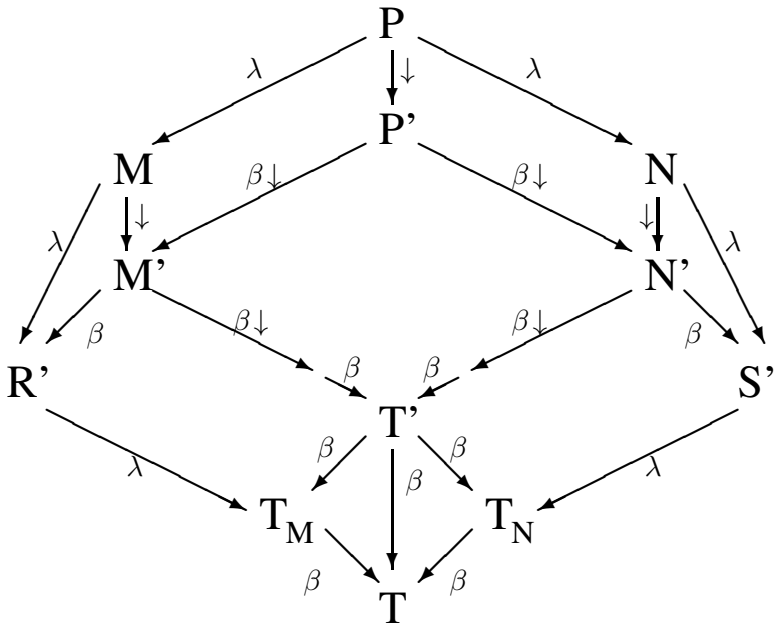

Figure 4: Schematic confluence of label-selective λ-calculus

PROOF For the numerical case, just take $\mathcal{S}$ with only one element.

For symbols, you just add the index 1 to all of them. No rule increases the index, and since all indices are equal, no rule decreases them. As a consquence any reduction on a symbolic selective λ-term in selective λ-calculus uses only steps of the symbolic calculus, and we get confluence by injection.

For the flat case, take $\mathcal{S}' = \mathcal{S} \cup \{\epsilon\}$, and define ϵ to be the least element of $\mathcal{S}'$. On the $\mathcal{S}$ part we can apply the argument for symbols, and get indices on the extra symbol ϵ. Rules (1'), (2'), (3') are ensured by the extended order. □

5 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 [13]. 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 [20] 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.

Based on our remarks of Section 3.3, we have formulated and are studying several concurrent calculi extending label-selective λ-calculus towards full concurrency [3], including the provision for computable channel names. One of the gains expected is that λ-calculus will need not be *encoded* as in [18], but directly embedded as syntactic identity.

Another application of this insight, in a pure confluent calculus, is studied as a new approach to state handling in the lambda calculus [10, 11]. In a composition-based system, similar in this respect to Categorical Combinatory Logic [7], we use labels as a way to select or modify directly the part of the state we are interested in. Some even stronger extension of the notion of currying towards binary relations can be done on this basis,

with a symmetric calculus of transformations [12].

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] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, Vol. 1, No. 4, pp. 375–416, October 1991.

[2] Hassan Aït-Kaci and Jacques Garrigue. Label-selective λ-calculus: Syntax and confluence. In *Proc. of the Conference on Foundations of Software Technology and Theoretical Computer Science*, pp. 24–40, Bombay, India, 1993. Springer-Verlag LNCS 761.

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

[4] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, Vol. 16, No. 3&4, pp. 195–234, July-August 1993.

[5] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *Proceedings of TAPSOFT '89*, pp. 149–161, Berlin, Germany, 1989. Springer-Verlag, LNCS 351.

[6] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indag. Math.*, Vol. 34, pp. 381–392, 1972.

[7] Pierre-Louis Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1993. First edition by Pitman, 1986.

[8] Laurent Dami. HOP: Hierarchical objects with ports. In D. Tsichritzis, editor, *Object Frameworks*. Technical report, University of Geneva, 1992.

[9] Laurent Dami. *Software Composition: Towards an Integration of Functional and Object-Oriented Approaches*. PhD thesis, Université de Genève, Faculté des Sciences Économiques et Sociales, Switzerland, 1994.

[10] Jacques Garrigue. Transformation calculus and its typing. In *Proc. of the workshop on Type Theory and its Applications to Computer Systems*, pp. 34–45. Kyoto University RIMS Lecture Notes 851, August 1993.

[11] Jacques Garrigue. The transformation calculus. Technical Report 94-09, University of Tokyo, Department of Information Science, April 1994.

[12] Jacques Garrigue. *Label-Selective Lambda-Calculi and Transformation Calculi*. PhD thesis, University of Tokyo, Department of Information Science, March 1995.

[13] Jacques Garrigue and Hassan Aït-Kaci. The typed polymorphic label-selective λ-calculus. In *Proc. ACM Symposium on Principles of Programming Languages*, pp. 35–47, 1994.

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

[15] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, Vol. 27, No. 4, pp. 797–821, October 1980.

[16] Peter J. Landin. The mechanical evaluation of expressions. *Computer Journal*, Vol. 6, No. 4, pp. 308–320, 1964.

[17] Henry Ledgard. *ADA : An Introduction, Ada Reference Manual (July 1980)*. Springer-Verlag, 1981.

[18] Robin Milner. Functions as processes. Rapport de Recherche 1154, INRIA, Rocquencourt, France, February 1990.

[19] Robin Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, pp. 203–246. NATO ASI Series, Springer Verlag, 1992.

[20] Atsushi Ohori. A compilation method for ML-style polymorphic records. In *Proc. ACM Symposium on Principles of Programming Languages*, pp. 154–165, 1992.

[21] Guy L. Steele. *Common LISP : The Language*. Digital Press, 1984.