A Rho-Calculus of explicit constraint application

Horatiu Cirstea, Germain Faure, Claude Kirchner

Noname manuscript No.
(will be inserted by the editor)

A ρ-calculus of explicit constraint application

Horatiu Cirstea[†] · **Germain Faure**[‡] · **Claude Kirchner**[*]

the date of receipt and acceptance should be inserted later

Abstract Theoretical presentations of the ρ-calculus often treat the matching constraint computations as an atomic operation although matching constraints are explicitly expressed. Actual implementations have to take a more realistic view: computations needed in order to find the solutions of a matching equation can have an important impact on the (efficiency of the) calculus for some matching theories and the substitution application usually involves a term traversal.

Following the works on explicit substitutions in the λ-calculus, we present two versions of the ρ-calculus, one with *explicit matching* and one with *explicit substitutions*, together with a version that combines the two and considers efficiency issues and more precisely the *composition* of substitutions. The approach is general, allowing for potential extensions to various matching theories. We establish the confluence of the calculus and the termination of the explicit constraint handling and application sub-calculus.

Keywords Rewriting calculus · explicit substitution · explicit matching · pattern matching.

1 Introduction

The ability to define functions by pattern matching is a powerful capability of languages like ELAN [8], Maude [17], Haskell [27] and ML [25,10]. Because of its induced programming and formal agility, matching is also a basic ingredient of formal islands over Java or C as implemented in the TOM system [40,34]. Furthermore, pattern matching is not only a useful programming paradigm, it can also deeply influence the computational behavior of a calculus, like its termination [16, 50]. The algorithms solving the corresponding matching problems can lead to the adequate solution(s) or can fail; but this latter information is usually not expressed explicitly in the above mentioned formalisms. This ability to express matching failures is a key point of the ρ-calculus.

The ρ-calculus was introduced to make all the basic ingredients of rewriting explicit objects, in particular the notions of rule, *i.e.* of *abstraction*, rule *application* and *result*. In the ρ-calculus, the usual λ-abstraction $\lambda X.N$ is generalized by a rule abstraction $P \twoheadrightarrow N$, where P is now an arbitrary term and not necessarily a variable X, and N is the argument to be fired. In such a rule, the free variables of P are bound in N. The application of a rule $P \twoheadrightarrow N$ to a term M, denoted $(P \twoheadrightarrow N)\ M$, evaluates to a term $\sigma(N)$ (the application of the substitution σ to the term N) where σ represents the solution(s) of the matching between P and M.

The matching power of the ρ-calculus can be adjusted by using arbitrary theories. In classical term rewriting, this can lead to non-deterministic behaviors (*e.g.* what is the result of the application of $(X+Y) \twoheadrightarrow X$ to $a+b$ if $+$ is commutative?) but, since "results" are first class citizens in the ρ-calculus, we can represent all possible results as a single one using the structure operator denoted by "$\wr$" (*e.g.* $a \wr b$). The way these results are represented is also a parameter of the calculus since different semantics are obtained according to the theories associated to the structure operator. Typically, if an associative-commutative and idempotent status is given to this operator then, we recover the semantics of result *sets* [13]. If one prefers lists or multisets, then the corresponding formalization should be specified.

† Université Nancy 2 & LORIA[††],
‡ Université Henri Poincaré & LORIA[††],
∗ INRIA & LORIA[††],
†† LORIA, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex France
E-mail: name.surname@loria.fr

The ability to parameterize the ρ-calculus by a matching theory opens new possibilities and leads to a very expressive calculus. Nevertheless, it is surprising that all the computations related to the considered matching theory still belong to the meta-level. The same situation arises in Rogue [49], a programming language based on an untyped version of the ρ-calculus and primarily intended for implementing decision procedures. The operational semantics of Rogue as well as the rules of the ρ-calculus use helper functions that are indeed implicit computations. These computations are conceptually and computationally important in all matching theories, from syntactic ones to quite elaborated ones like associative-commutative theories [23]. Therefore, to make explicit the handling of constraints, one must make explicit the matching and the constraint (substitution) application.

A first step toward an explicit handling of the matching related computations was the introduction of matching problems as part of the ρ-calculus syntax [15]. More precisely, the *matching constraints* represent constrained terms which are eventually instantiated by the substitution obtained as solution of the corresponding matching problem (if such a solution exists). Matching failures can be treated in different ways. In early versions of the calculus [14] the application of a rule abstraction involving a matching failure reduced to a special term representing *the failure* term. Alternatively, in the current version of the calculus, the matching constraints where the corresponding matching problem has no solution are in normal form.

In both cases, the matching constraints are solved and the resulting substitutions are applied in one step. In concrete implementations these operations should be separated and should interact with other computations and, in particular, we want computations on constraints and applications of constraints to be explicit.

The evaluation rules solving the syntactic matching problems at the object level were proposed in [12]. These rules eventually transform a term constrained by a matching problem to the same term but constrained by the solved problem, that is, by a substitution. The application of the resulting substitutions follows the approaches used in λ-calculi with explicit substitutions.

These calculi have been widely studied and provide a nice tool to deal with higher-order unification [22] or to represent incomplete proofs in type theory [41]. As far as implementation issues are concerned, explicit substitution calculi play a central role in some implementations of ML [38].

In all the explicit substitution calculi [1,37,46], substitutions can be delayed thanks to the β rule that transforms a β-redex $(\lambda x.a)b$ into the *explicit* application on a of the substitution that replaces x by b. In the explicit ρ-calculus, the application of substitutions is delayed to the moment where the original matching constraint is solved. Thus, the role of the β rule is taken by the ρ rule which transforms the application of a rewrite rule into the application of a matching constraint and by the evaluation rules solving the obtained matching problem. This led to a calculus *à la* λ_x-calculus [46] simple and without substitution compositions that we called ρ_x.

This calculus [12] handles at the object level not only the matching problems but also the application of the resulted substitutions. Nevertheless, these two computations are handled differently and considered one at a time. In this paper we present two versions of the ρ-calculus, one with explicit matching and one with explicit substitutions, together with a version that combines the two and considers efficiency issues and more precisely the composition of substitutions. This allows us to isolate the features absolutely necessary in both cases and to analyze the issues related to the two approaches. The result is a full calculus that enjoys the usual good properties of explicit substitutions (conservativity, termination) and which is confluent. We show that the ρ-calculus, and especially explicit ρ-calculi, are suitable as a useful theoretical back-end for implementations.

The paper is organized as follows. Sections 2 and 3 consider respectively simple extensions of the plain ρ-calculus for explicit substitution ($\rho_{\mathtt{s}}$) and explicit matching ($\rho_{\mathtt{m}}$). Section 4 first presents the combination (ρ_x°) of the two previous calculi enriched with a rule for combining term traversal (composition rule for substitutions). Secondly, properties such as the confluence of the calculus and the termination of the explicit part are then established. Some possible extensions are proposed and briefly discussed. We finally give in Section 5 some hints on a direct implementation of explicit ρ-calculi.

2 Explicit substitution

We introduce in this section a generalization of the λ_x-calculus [46] that we call ρ-calculus with explicit substitutions and that considers abstractions not only on single variables but also on patterns potentially containing several variables. On the other hand, the obtained ρ-calculus with explicit substitutions, denoted shortly $\rho_{\mathtt{s}}$, can be seen as an extension of the plain ρ-calculus with explicit substitutions.

In the plain ρ-calculus, when reducing the application of a rule to a term, the matching between the left-hand side of the rule and the term is solved and the resulting substitution is applied to the right-hand side of the rule at the meta-level of the calculus. This means that, in one step, we compute the substitution solving the corresponding matching problem and apply it.

Terms	M,N,P	$::= X$	(Variables)
		$\mid c$	(Constants)
		$\mid P \twoheadrightarrow M$	(Abstraction)
		$\mid M\,N$	(Functional application)
		$\mid M \wr N$	(Structure)
		$\mid N[\phi]$	(Substitution application)
Substitutions	ϕ	$::= \mathtt{id}$	(Identity)
		$\mid \bigwedge_{i=1\ldots n} X_i = M_i$	(Conjunction)
		where $\wedge$ is associative	

Fig. 1 Syntax of $\rho_{\mathtt{s}}$

This reduction can be obviously decomposed into two steps, one computing the substitution and the other one describing the application of this substitution. This decomposition does not mean that the matching related computations and the application of substitutions are explicit but just that they are clearly separated. In this section we go a step further toward an explicit version of the ρ-calculus by proposing a version of the calculus where the substitution application is performed explicitly while the matching problems are still solved at the meta-level.

2.1 Syntax of $\rho_{\mathtt{s}}$

Since we focus here only on explicit substitution application and not on explicit matching, the general syntax of $\rho_{\mathtt{s}}$ presented in Figure 1 restricts the *matching constraints* of the form $N[P \ll M]$ from the plain ρ-calculus [14] to *substitution applications* of the form $N[X = M]$. More precisely, the general patterns of the matching constraints are restricted to simple variables. This means that the matching constraints that we handle here are always in solved form and thus, the corresponding substitution can be (explicitly) applied.

As in the plain ρ-calculus, the left-hand side of an *abstraction* (built using the "$\twoheadrightarrow$" operator) defines the variables we abstract on and some context information. A term in a left-hand side of an abstraction is called a *pattern*. An *application* is implicitly denoted by concatenation. The terms can be grouped together into *structures* (built using the operator "$\wr$"). For the scope of this paper we restrict to syntactic structures, *i.e.* no theory is associated to the structure operator.

The *substitution application* operator is a generalization of the similar one from λ_x-calculus. In the λ-calculus, a λ-abstraction binds only one variable and thus an explicit substitution consists in a single variable binding. In the ρ-calculus, the pattern-abstraction can bind an arbitrary number of variables and thus the definition of a substitution is extended to support multiple bindings. The `id` symbol represents the identity substitution. We should point out that in this context the symbol "=" is not symmetric.

We assume that the functional and substitution application operators associate to the left, while the other operators associate to the right. The priority of the substitution application is higher than that of the functional application which is higher than that of "$\_ \twoheadrightarrow \_$" which is, in turn, of higher priority than the "$\_ \wr \_$".

The symbols $M,N,\ldots$ range over the set $\mathscr{T}$ of terms, the symbols $X,Y,Z,\ldots$ range over the set $\mathscr{V}$ of variables ($\mathscr{V} \subseteq \mathscr{T}$), the symbols $a,b,c,\ldots,f,g,h$ range over a set $\mathscr{K}$ of term constants ($\mathscr{K} \subseteq \mathscr{T}$). Finally, the symbols P,Q range over the set $\mathscr{P}$ of patterns, ($\mathscr{V} \subseteq \mathscr{P} \subseteq \mathscr{T}$). We call *algebraic* the terms of the form $(\ldots((f\,A_1)\,A_2)\ldots)\,A_n$ with $f \in \mathscr{K}$ and we usually denote them by $f(A_1,A_2,\ldots,A_n)$. The symbols $\phi,\psi,\ldots$ range over the set Φ of substitutions. A term is called *pure* if it does not contain any explicit substitution application.

For the purpose of this paper we restrict to patterns that are either algebraic terms or structures consisting of this kind of terms:

Patterns	$P,Q ::= X \mid c \mid f(P_1,P_2,\ldots,P_n) \mid P \wr Q$

To simplify the reading, we adopt the following notation

$$N[X_i = M_i]_{i=1}^n \triangleq \begin{cases} N\left[\bigwedge_{i=1\ldots n} X_i = M_i\right] & \text{if } n > 0 \\ \mathtt{id} & \text{otherwise} \end{cases}$$

The *domain* of a substitution $\phi = \bigwedge_{i=1\ldots n} X_i = M_i$, denoted $\mathscr{D}om(\phi)$, is the set $\{X_i\}_{i=1}^n$.

Since we work in a calculus with binders, the usual notions of free and bound variables from λ-calculus are naturally extended by considering that all variables in the left-hand side of an abstraction are bound in the abstraction. The formal definition for the set of free variables for the fore-coming ρ°_{x} calculus is given in Section 4 and the restriction to the ρ-calculus with explicit substitutions is straightforward. As in any calculus involving binders, we work modulo the α-*conversion* of Church, and modulo the *hygiene-convention* of Barendregt [2], *i.e.*, free and bound variables have different names.

Remark 1 (Translation of λ-terms)
One can encode the λ-calculus in the ρ-calculus: given the set of λ-terms defined by

$$M,N ::= X \mid \lambda X.M \mid MN$$

the translation function $[\![\ ]\!]$ from λ-terms to ρ-terms is defined by

$$\begin{array}{ll} [\![X]\!] & = X \\ [\![\lambda X.M]\!] & = X \twoheadrightarrow [\![M]\!] \\ [\![MN]\!] & = [\![M]\!]\,[\![N]\!] \end{array}$$

Thus, when translating λ-terms into ρ-terms the binder "λ" is replaced by the (rule) abstraction operator "$\twoheadrightarrow$" like for the following terms:

λ-calculus	ρ-calculus
$\lambda X.X$	$X \twoheadrightarrow X$
$\lambda X.\lambda Y.X$	$X \twoheadrightarrow Y \twoheadrightarrow X$
$\lambda X.(XX)$	$X \twoheadrightarrow XX$

As we will see in the next section, this translation is consistent with the reduction in the two formalisms: for each β-reduction of a λ-term there exists a corresponding reduction of the translated term in the ρ-calculus.

Example 1 (Encoding of propositional formulae)
Using the constants true, false, not, and, or (denoting respectively the boolean values true and false, the negation, the conjunction and the disjunction) we can define the following propositional formulae: $\mathsf{and}(X,\mathsf{true})$ and $\mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))$.

Example 2 (Rewrite rules)
Some rules to compute in the Boolean algebra:

- $\mathsf{and}(X,\mathsf{true}) \twoheadrightarrow X$; the variable X is free in the pattern $\mathsf{and}(X,\mathsf{true})$ and bound in the body X of the abstraction.
- $\mathsf{not}(\mathsf{and}(X,Y)) \twoheadrightarrow \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))$; this rule bounds the variables X and Y.
- $\mathsf{xor}(X,X) \twoheadrightarrow \mathsf{false}$; a non-linear rule.

The application of the second rewrite rule to the term $\mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false}))$ is represented by the term $\big(\mathsf{not}(\mathsf{and}(X,Y)) \twoheadrightarrow \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\big)\ \mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false}))$ and, as we will see in the next section, this term reduces to $\mathsf{or}(\mathsf{not}(\mathsf{true}),\mathsf{not}(\mathsf{false}))$.

2.2 Operational semantics of $\rho_{\mathtt{s}}$

The evaluation mechanism of the calculus relies on the fundamental operation of *matching* that allows us to bind variables to their current values. Since we want to define an expressive and powerful calculus, we allow the matching to be performed *modulo* a congruence on terms. This congruence used at matching time is a fundamental parameter of the calculus and different instances are obtained when instantiating this parameter by a congruence defined, for example, syntactically, or equationally or in a more elaborated way [14].

Definition 1 (Matching)
Given a theory $\mathbb{T}$ (*i.e.* a set of axioms defining a congruence relation $\overset{\mathbb{T}}{=}$):

1. A *matching equation* is a problem $P \prec\!\!< A$ with P a pattern and A a term.
2. A substitution θ is a solution of the matching equation $P \prec\!\!< A$ if $\theta P \overset{\mathbb{T}}{=} A$.

The set of solutions of $P \prec\!\!< A$ is denoted by $\mathscr{S}ol(P \prec\!\!< A)$.

$$
\begin{array}{lllll}
(\rho) & (P \twoheadrightarrow M)\ N & \to & M\left[X_i = Q_i\right]_{i=1}^{n} & \text{where } \mathscr{S}ol(P \lessdot N) = \{Q_i/X_i\}_{i=1}^{n} \\
(\delta) & (M_1 \wr M_2)\ N & \to & M_1\ N \wr M_2\ N & \\
(Identity) & M[\mathtt{id}] & \to & M & \\
(Replace) & X[\phi \wedge (X = M) \wedge \psi] & \to & M & \\
(Var) & Y[\phi] & \to & Y & Y \notin \mathscr{D}om(\phi) \\
(Const) & c[\phi] & \to & c & \\
(Abs) & (P \twoheadrightarrow M)[\phi] & \to & P[\phi] \twoheadrightarrow M[\phi] & \\
(App) & (M\ N)[\phi] & \to & M[\phi]\ N[\phi] & \\
(Struct) & (M \wr N)[\phi] & \to & M[\phi] \wr N[\phi] &
\end{array}
$$

Fig. 2 Small-step operational semantics of $\rho_{\mathtt{s}}$

When restricting to syntactic matching, the matching substitution, when it exists, is unique and can be computed by a simple recursive algorithm given for example by G. Huet [28]. It can also be computed by the following set of rules, that are applied modulo the associativity and commutativity of the symbol $\wedge$. As quite natural for a matching problem, we assume the sets of variables of the left-hand and right-hand side of the equation to be disjoint. Otherwise, we refer to [32] for a complete set of transformation rules.

$$
\begin{array}{lcll}
M_1 \wr M_2 \lessdot N_1 \wr N_2 & \to & M_1 \lessdot N_1 \wedge\ M_2 \lessdot N_2 & \\
f(M_1,\ldots,M_n) \lessdot f(N_1,\ldots,N_n) & \to & \wedge_{i=1}^{n}(M_i \lessdot N_i) & (n \geq 1) \\
X \lessdot M \quad \wedge \quad X \lessdot M & \to & X \lessdot M & \\
f(M_1,\ldots,M_n) \lessdot g(N_1,\ldots,N_m) & \to & \mathbb{F} & \text{if } f \neq g \\
M \lessdot X & \to & \mathbb{F} & \text{if } M \neq X \\
X \lessdot M \quad \wedge \quad X \lessdot N & \to & \mathbb{F} & \text{if } M \neq N \\
\mathbb{F} \quad \wedge \quad C & \to & \mathbb{F} &
\end{array}
$$

Starting from a matching equation, the application of this rule set terminates and returns either $\mathbb{F}$ when there are no substitutions solving the equation, or a conjunction of match equations $\wedge_{i=1}^{n}(X_i \lessdot Q_i) \wedge_{j=1}^{m}(c_j \lessdot c_j)$ in "normal form" from which the solution $\{Q_i/X_i\}_{i=1}^{n}$ can be trivially inferred [32]. This set of rules could be extended to deal with more elaborated theories like commutativity.

The operational semantics of the ρ-calculus with explicit substitutions is given in Figure 2 where the reduction rules of the calculus are split into two categories:

- Rules describing the application of structures and abstractions on ρ-terms.
- Rules defining the application of substitutions.

The (ρ) rule is used to reduce the application of an abstraction to a term by matching the left-hand side of the abstraction against the term and triggering the application of the obtained substitution to the right-hand side of the abstraction. If no match exists, the rule is not applied. The rule (δ) is inherited from the plain ρ-calculus and deals with the distributivity of the application on the structures built with the "$\wr$" operator.

The rules handling the substitution application distribute it over the different operators until a variable or a constant is reached. If the variable is in the domain of the substitution then the corresponding term replaces it; a substitution applied to a variable that is not in its domain or to a constant is ignored. Since we consider classes of terms modulo α-conversion, the appropriate representatives are always chosen in order to avoid potential variable captures (introduced by the rule (Abs)).

In this paper we consider a matching theory that is supposed to be decidable and unitary. These restrictions allow us to focus on the design aspects of the ρ-calculus and to have a modular approach to the intrinsic matching algorithms. Non-unitary matching theories (*e.g.* equational) can be also considered but the meta-theory is more complicated in this case. More general patterns can be also used and the possible higher-order matching that should be used in this case is under investigation.

We denote, as usually [2], the compatible closure of a relation $\mathscr{R}$ by $\mapsto_{\mathscr{R}}$ (also denoted $\mathscr{R}$) and the transitive closure of $\mapsto_{\mathscr{R}}$ by $\mapsto\!\!\!\!\to_{\mathscr{R}}$ (also denoted $\mathscr{R}^*$). This way we define the relations $\mapsto_\sigma$ and $\mapsto_{\rho_\mathsf{s}}$ induced by the rules dealing with substitutions (*i.e.* (*Identity*), (*Replace*), (*Var*), (*Const*), (*Abs*), (*App*), (*Struct*)) and by the set of all rules in Figure 2, respectively. We should point out that in ρ_s and in the other calculi introduced in the next sections all the evaluation steps are performed modulo the underlying theory for the conjunction (see Definition 3 for a precise definition of rewriting modulo).

One can notice that when replacing the pattern P by a variable in rule (ρ), we recover the (*Beta*) rule of λ-calculi with explicit substitution.

Example 3 (Application of a rewrite rule)
In order to compute the disjunctive normal form of a propositional formula, we use the rewrite rule $\mathsf{not}(\mathsf{and}(X,Y)) \to \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))$. The application of this rewrite rule to the term $\mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false}))$ is described in the ρ-calculus by the following reduction:

$$
\begin{array}{ll}
 & \big(\mathsf{not}(\mathsf{and}(X,Y)) \to \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\big)\ \mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false})) \\
\mapsto_{\rho_\mathsf{s}} & \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\,[X=\mathsf{true}\wedge Y=\mathsf{false}] \\
\mapsto\!\!\!\!\to_{App} & (\mathsf{or}\,[X=\mathsf{true}\wedge Y=\mathsf{false}])(\mathsf{not}(X)\,[X=\mathsf{true}\wedge Y=\mathsf{false}],\mathsf{not}(Y)\,[X=\mathsf{true}\wedge Y=\mathsf{false}]) \\
\mapsto_{Const} & \mathsf{or}(\mathsf{not}(X)\,[X=\mathsf{true}\wedge Y=\mathsf{false}],\mathsf{not}(Y)\,[X=\mathsf{true}\wedge Y=\mathsf{false}]) \\
\mapsto\!\!\!\!\to_{\sigma} & \mathsf{or}(\mathsf{not}(X\,[X=\mathsf{true}\wedge Y=\mathsf{false}]),\mathsf{not}(Y\,[X=\mathsf{true}\wedge Y=\mathsf{false}])) \\
\mapsto\!\!\!\!\to_{Replace} & \mathsf{or}(\mathsf{not}(\mathsf{true}\,[Y=\mathsf{false}]),\mathsf{not}(Y\,[X=\mathsf{true}\wedge Y=\mathsf{false}])) \\
\mapsto\!\!\!\!\to_{Var} & \mathsf{or}(\mathsf{not}(\mathsf{true}\,[\mathtt{id}]),\mathsf{not}(Y\,[X=\mathsf{true}\wedge Y=\mathsf{false}])) \\
\mapsto\!\!\!\!\to_{Identity} & \mathsf{or}(\mathsf{not}(\mathsf{true}),\mathsf{not}(Y\,[X=\mathsf{true}\wedge Y=\mathsf{false}])) \\
\mapsto\!\!\!\!\to_{\sigma} & \mathsf{or}(\mathsf{not}(\mathsf{true}),\mathsf{not}(\mathsf{false}))
\end{array}
$$

Example 4 (Application of a rewrite system)
We show how a structure of rewrite rules applies to a term. In a first approximation, this can be seen as the application of a rewrite system.

$$\Big(\mathsf{and}(X,\mathsf{or}(Y,Z)) \to \mathsf{or}(\mathsf{and}(X,Y),\mathsf{and}(X,Z)) \wr\ \mathsf{and}(\mathsf{or}(X,Y),Z) \to \mathsf{or}(\mathsf{and}(X,Z),\mathsf{and}(Y,Z))\Big)\ \mathsf{and}(\mathsf{or}(\mathsf{true},\mathsf{false}),\mathsf{or}(\mathsf{false},\mathsf{false}))$$

We use the (δ) rule to distribute the two rewrite rules:

$$
\begin{array}{ll}
\mapsto_{\delta} & \big(\mathsf{and}(X,\mathsf{or}(Y,Z)) \to \mathsf{or}(\mathsf{and}(X,Y),\mathsf{and}(X,Z))\big)\ \mathsf{and}(\mathsf{or}(\mathsf{true},\mathsf{false}),\mathsf{or}(\mathsf{false},\mathsf{false})) \wr \\
 & \big(\mathsf{and}(\mathsf{or}(X,Y),Z) \to \mathsf{or}(\mathsf{and}(X,Z),\mathsf{and}(Y,Z))\big)\ \mathsf{and}(\mathsf{or}(\mathsf{true},\mathsf{false}),\mathsf{or}(\mathsf{false},\mathsf{false}))
\end{array}
$$

and we finally obtain (by reducing each rule application as in Example 3):

$$\mathsf{or}(\mathsf{and}(\mathsf{or}(\mathsf{true},\mathsf{false}),\mathsf{false}),\mathsf{and}(\mathsf{or}(\mathsf{true},\mathsf{false}),\mathsf{false}))\ \wr\ \mathsf{or}(\mathsf{and}(\mathsf{true},\mathsf{or}(\mathsf{false},\mathsf{false})),\mathsf{and}(\mathsf{false},\mathsf{or}(\mathsf{false},\mathsf{false})))$$

The application of a rewrite system is actually never as simple as presented above. Here, we encode only one (meta) rewriting step but, in general, the encoding is more complicated because one needs to encode not only the application of a rewrite rule at the top position of a term but also the reduction strategy guiding the application of the rules. The problem can be solved, for example, by using (typed) fix-points to apply the rewrite system recursively (see [16] for a full presentation).

Example 5 (Multiple applications)
Since substitutions cannot be composed, the application of each substitution is done independently and can involve a lot of evaluation steps needed in order to access to the leaves of the term (seen as a tree).

$$
\begin{array}{ll}
 & (g(X) \to ((f(Y) \to h(X,Y))\ f(a))\ g(b) \\
\mapsto_{\rho} & ((f(Y) \to h(X,Y))\ f(a))\,[X=b] \\
\mapsto_{\rho} & h(X,Y)\,[Y=a]\,[X=b] \\
\mapsto\!\!\!\!\to_{\sigma} & h(X,Y\,[Y=a])\,[X=b] \\
\mapsto\!\!\!\!\to_{\sigma} & h(X,a)\,[X=b] \\
\mapsto\!\!\!\!\to_{\sigma} & h(X\,[X=b],a) \\
\mapsto\!\!\!\!\to_{\sigma} & h(b,a)
\end{array}
$$

If the substitutions are composed, then the term traversal related steps can be factorized as shown in Example 10.

3 Explicit matching

In this section we concentrate on the matching problems intrinsic to the ρ-calculus and, more precisely, we want to make explicit the matching computations performed during the ρ-evaluation. We propose here the ρ-calculus with explicit match-

Terms	M,N,P	$::= X$	(Variables)
		$\mid c$	(Constants)
		$\mid P \twoheadrightarrow M$	(Abstraction)
		$\mid M\ N$	(Functional application)
		$\mid M \wr N$	(Structure)
		$\mid N[\mathscr{C}]$	(Constraint application)
Constraints	$\mathscr{C},\mathscr{D}$	$::= \mathtt{id}$	(Identity)
		$\mid P \ll M$	(Match-equation)
		$\mid \mathscr{C} \wedge \mathscr{D}$	(Conjunction of constraints)
		where $\wedge$ is associative and $\mathtt{id}$ is a neutral element	

Fig. 3 Syntax of $\rho_{\mathtt{m}}$

ing, denoted also $\rho_{\mathtt{m}}$, that extends the plain ρ-calculus with evaluation rules dealing with the (syntactic) matching. In this calculus the obtained matching problems are solved explicitly while the substitution application is done at the meta-level.

3.1 Syntax of $\rho_{\mathtt{m}}$

The syntax of the ρ-calculus with explicit matching is given in Figure 3. The explicit substitutions of $\rho_{\mathtt{s}}$ that can be considered as match equations in solved form are replaced by match constraints that will be solved in the evaluation process.

The symbols $\mathscr{C},\mathscr{D},\mathscr{E}\ldots$ range over the set of (possibly empty) constraints. The $\mathtt{id}$ symbol represents here the identity constraint while in the $\rho_{\mathtt{s}}$ the same symbol was used to represent the empty substitution.

The domain of a constraint $\mathscr{C}$, denoted $\mathscr{D}om(\mathscr{C})$, is intuitively the same as the domain of the substitution that solves all the corresponding matching problems and is computed by taking the union of the sets of free variables of the patterns of all the match-equations in $\mathscr{C}$. The formal definition is given in Section 4.1.

3.2 Operational semantics of $\rho_{\mathtt{m}}$

The operational semantics of the ρ-calculus with explicit matching consists of two parts as shown in Figure 4.

As for $\rho_{\mathtt{s}}$, the first part describes the application of structures and abstractions on ρ-terms. This time the (ρ) rule always applies and reduces the application of an abstraction to a term constraint by the corresponding matching problem.

The matching problems are handled by the second part of the evaluation rules that are clearly inspired by the ones presented in Section 2.2. A matching constraint is simplified using the decomposition rules which are strongly related to the considered matching theory. As we have already mentioned, we consider only structures of algebraic terms as patterns and we restrict to a decidable and unitary matching theory and, more precisely, to syntactic matching. Therefore, we do not handle the higher-order symbols (*e.g.* "$\twoheadrightarrow$", "$\ll$") and we only decompose the restricted patterns. The two decomposition rules given in Figure 4 are thus performed *w.r.t.* to an empty matching theory for the structure operator and for the constant symbols. We consider that an empty conjunction and $\mathtt{id}$ represent the same object and thus, the rule $(\mathit{Decompose}_{\mathscr{F}})$ can be used for constants (*i.e.* $n=0$) in which case the result is the identity.

When a part of the constraint is solved and independent of the rest of the constraint, the corresponding substitution can be applied at the meta-level. We consider that the meta-application of the substitution $\{M/X\}$ to the term N, denoted $N\{M/X\}$, is higher-order and thus performs α-conversion in order to avoid the possible variable captures. The condition in rule $(\mathit{ToSubst})$ guarantees that a matching problem is solved and thus (part of) the corresponding substitution can be applied only if all its variables are assigned the same term. Non-linear matching problems can lead to matching constraints which assign different terms to the same variable and which represent, intuitively, a failure (see Example 8). The doubletons in a matching constraint are eliminated with the rule (Idem).

Notice that in the rule $(\mathit{ToSubst})$, because of the hygiene-convention, the intersection between the set of free variables of M and $\mathscr{D}om(\mathscr{C}\wedge\mathscr{D})$ is empty and thus, the variables in M cannot be captured in the right-hand side of the rule.

The ρ-calculus (and in particular $\rho_{\mathtt{m}}$) is well-suited to deal with (matching) errors, represented by constraints without solution, that is, constraints that do not represent substitutions. Depending on the intended use of the calculus we may want or not to propagate such (constraint) failures. If the failures are propagated, the error's location is lost and the final result would be a term with constraints with no solution applied on each leaf of the term (considered as a tree). The information contained in such a term seems useless when one wants to analyze the error and, for debugging reasons, we do not want to lose the error's location. This is why the failures are not propagated as they are but only the corresponding substitution (if one exists) is propagated.

$$
\begin{array}{lll}
(\rho) & (P \twoheadrightarrow M)\ N & \to\ M[P \ll N] \\
(\delta) & (M_1 \wr M_2)\ N & \to\ M_1\ N \wr M_2\ N \\
(\mathit{Decompose}_\iota) & M_1 \wr M_2 \ll N_1 \wr N_2 & \to\ M_1 \ll N_1 \wedge M_2 \ll N_2 \\
(\mathit{Decompose}_{\mathscr{F}}) & f(M_1,\ldots,M_n) \ll f(N_1,\ldots,N_n) & \to\ \bigwedge_{i=1}^{n} (M_i \ll N_i) \\
(\mathit{Idem}) & \mathscr{C} \wedge (X \ll M) \wedge \mathscr{D} \wedge (X \ll M) \wedge \mathscr{E} & \to\ \mathscr{C} \wedge (X \ll M) \wedge \mathscr{D} \wedge \mathscr{E} \\
(\mathit{ToSubst}) & N[\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D}] & \to\ (N\{M/X\})[\mathscr{C} \wedge \mathscr{D}] \\
 & & \quad \text{if } X \notin \mathscr{D}om(\mathscr{C} \wedge \mathscr{D}) \\
(\mathit{Identity}) & M[\mathtt{id}] & \to\ M
\end{array}
$$

Fig. 4 Small-step operational semantics of $\rho_{\mathtt{m}}$

Example 6 (Application of a rewrite rule)
The term presented in Example 3 reduces in ρ-calculus with explicit matching as follows:

$$
\begin{array}{ll}
 & \big(\mathsf{not}(\mathsf{and}(X,Y)) \twoheadrightarrow \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\big)\ \mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false})) \\
\mapsto_{\rho} & \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\,[\mathsf{not}(\mathsf{and}(X,Y)) \ll \mathsf{not}(\mathsf{and}(\mathsf{true},\mathsf{false}))] \\
\mapsto\!\!\!\!\to_{\mathit{Decompose}_{\mathscr{F}}} & \mathsf{or}(\mathsf{not}(X),\mathsf{not}(Y))\,[X \ll \mathsf{true} \wedge Y \ll \mathsf{false}] \\
\mapsto\!\!\!\!\to_{\mathit{ToSubst}} & \mathsf{or}(\mathsf{not}(\mathsf{true}),\mathsf{not}(\mathsf{false}))\,[\mathtt{id}] \\
\mapsto_{\mathit{Identity}} & \mathsf{or}(\mathsf{not}(\mathsf{true}),\mathsf{not}(\mathsf{false}))
\end{array}
$$

Example 7 (Successful application of a non-linear rewrite rule)
When non-linear patterns are used, the rule (Idem) can merge the solved matching problems that are identical.

$$
\begin{array}{ll}
 & (\mathsf{xor}(X,X) \twoheadrightarrow \mathsf{false})\ \mathsf{xor}(\mathsf{true},\mathsf{true}) \\
\mapsto_{\rho} & \mathsf{false}\,[\mathsf{xor}(X,X) \ll \mathsf{xor}(\mathsf{true},\mathsf{true})] \\
\mapsto_{\mathit{Decompose}_{\mathscr{F}}} & \mathsf{false}\,[X \ll \mathsf{true} \wedge X \ll \mathsf{true}] \\
\mapsto_{\mathit{Idem}} & \mathsf{false}\,[X \ll \mathsf{true}] \\
\mapsto\!\!\!\!\to_{\mathit{ToSubst}} & \mathsf{false}\,[\mathtt{id}] \\
\mapsto_{\mathit{Identity}} & \mathsf{false}
\end{array}
$$

Of course, the application of a non-linear rewrite rule may lead to failures due to merging clashes. Merging clashes are not reduced but kept as a constraint application failure.

Example 8 (Application of a non-linear rewrite rule with failure)

$$
\begin{array}{ll}
 & (\mathsf{xor}(X,X) \twoheadrightarrow \mathsf{false})\ \mathsf{xor}(\mathsf{true},\mathsf{false}) \\
\mapsto_{\rho} & \mathsf{false}\,[\mathsf{xor}(X,X) \ll \mathsf{xor}(\mathsf{true},\mathsf{false})] \\
\mapsto_{\mathit{Decompose}_{\mathscr{F}}} & \mathsf{false}\,[X \ll \mathsf{true} \wedge X \ll \mathsf{false}]
\end{array}
$$

The next example illustrates the usefulness of explicit matching when we want to track the source (cause) of the failure.

Example 9 (Run-time error: matching failure)
Let us consider the following rule that checks if two persons are brothers, *i.e.*, if they have the same father:

$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \twoheadrightarrow \mathtt{true}$

When checking if two concrete persons (*Alice* and *Bob*) are brothers by applying this rule to the corresponding term:

$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(\mathit{Alice}),\mathtt{Father}(\mathit{John})),\ \mathtt{Person}(\mathtt{Name}(\mathit{Bob}),\mathtt{Father}(\mathit{Jim})))$

we obtain as result the term

$\mathtt{true}\,[Z \ll \mathit{John} \wedge Z \ll \mathit{Jim}]$

indicating that the variable Z corresponding to the father cannot be instantiated correctly, *i.e.*, that the father of the two persons is not the same.

This is in contrast with the ρ-calculus with explicit substitutions where the application of the rule to the term is in normal form indicates that the matching has no solution but gives no information on the source of this failure.

Terms	M,N,P	$::= X$	(Variables)
		$\mid c$	(Constants)
		$\mid P \twoheadrightarrow M$	(Abstraction)
		$\mid M\,N$	(Functional application)
		$\mid M \wr N$	(Structure)
		$\mid N[\phi]$	(Substitution application)
		$\mid N[\mathscr{C}]$	(Constraint application)
Substitutions	ϕ,ψ	$::= id_s$	(Identity)
		$\mid X = M$	(Equation)
		$\mid \phi \wedge \psi$	(Conjunction of equations)
Constraints	$\mathscr{C},\mathscr{D}$	$::= id_m$	(Identity)
		$\mid P \ll M$	(Match-equation)
		$\mid \mathscr{C} \wedge \mathscr{D}$	(Conjunction of constraints)
		where $\wedge$ is associative and id_s and id_m are neutral elements	

Fig. 5 Syntax of ρ°_{x}

4 Explicit substitution and explicit matching

The combination of the two previously introduced calculi, $\rho_{\mathtt{s}}$ and $\rho_{\mathtt{m}}$, leads to a version of the calculus that handles explicitly the matching constraints resolution as well as the application of the substitutions. This calculus, called ρ_{x} and introduced in [12] does not handle the composition of substitutions, a key issue when one wants to obtain efficient implementations.

In what follows we add this feature and define ρ°_{x}. The properties of this calculus are then studied.

4.1 Syntax of ρ°_{x}

The syntax presented in Figure 5 merges the ones of $\rho_{\mathtt{s}}$ and $\rho_{\mathtt{m}}$ and defines ρ°_{x}-terms. In what follows we refer to the three categories of ρ°_{x}-terms by simply calling them *terms*, *substitutions* and *constraints* respectively.

One can notice that the conjunction operator $\wedge$ is overloaded and it is used to build substitutions as well as constraints. Since the two types of conjunctions are disjoint, in what follows, we will generally denote the corresponding identities id_s and id_m by the same symbol $\mathtt{id}$.

We assume that the functional, substitution and constraint application operators associate to the left, while the other operators associate to the right. The priority of the substitution application is higher than that of the constraint application which is higher than that of the functional application. The application has a higher priority than "$\_ \twoheadrightarrow \_$" which is, in turn, of higher priority than the "$\_ \wr \_$". The equation operators are of higher priority than the conjunction operators. The equation and conjuntion operators have a lower priority than the other ones.

Definition 2 (Free variables and constraint domains)
The set of free variables and the domain of a constraint (resp. substitution) are defined by:

$$
\begin{array}{lll}
\mathscr{FV}(X) = \{X\} & & \\
\mathscr{FV}(c) = \emptyset & & \mathscr{FV}(P \twoheadrightarrow M) = \mathscr{FV}(M) \backslash \mathscr{FV}(P) \\
\mathscr{FV}(M\,N) = \mathscr{FV}(M) \cup \mathscr{FV}(N) & & \mathscr{FV}(M \wr N) = \mathscr{FV}(M) \cup \mathscr{FV}(N) \\
\mathscr{FV}(N[\mathscr{C}]) = \mathscr{FV}(\mathscr{C}) \cup \big(\mathscr{FV}(N) \backslash \mathscr{Dom}(\mathscr{C})\big) & & \mathscr{FV}(N[\phi]) = \mathscr{FV}(\phi) \cup \big(\mathscr{FV}(N) \backslash \mathscr{Dom}(\phi)\big) \\
\\
\mathscr{FV}(\mathscr{C} \wedge \mathscr{D}) = \mathscr{FV}(\mathscr{C}) \cup \mathscr{FV}(\mathscr{D}) & & \mathscr{FV}(\phi \wedge \psi) = \mathscr{FV}(\phi) \cup \mathscr{FV}(\psi) \\
\mathscr{FV}(X = M) = FV(M) & & \mathscr{FV}(\mathtt{id}) = \emptyset \\
\mathscr{FV}(P \ll M) = \mathscr{FV}(M) & & \\
\\
\mathscr{Dom}(P \ll M) = \mathscr{FV}(P) & & \mathscr{Dom}(\mathscr{C} \wedge \mathscr{D}) = \mathscr{Dom}(\mathscr{C}) \cup \mathscr{Dom}(\mathscr{D}) \\
\mathscr{Dom}(\mathtt{id}) = \emptyset & & \\
\\
\mathscr{Dom}(X = M) = \{X\} & & \mathscr{Dom}(\phi \wedge \psi) = \mathscr{Dom}(\phi) \wedge \mathscr{Dom}(\psi)
\end{array}
$$

(ρ)	$(P \multimap M)\, N$	$\to$	$M[P \ll N]$
(δ)	$(M_1 \wr M_2)\, N$	$\to$	$M_1\, N \wr M_2\, N$
$(Decompose_\iota)$	$M_1 \wr M_2 \ll N_1 \wr N_2$	$\to$	$M_1 \ll N_1 \wedge M_2 \ll N_2$
$(Decompose_\mathscr{F})$	$f(M_1,\ldots,M_n) \ll f(N_1,\ldots,N_n)$	$\to$	$\bigwedge_{i=1}^{n} (M_i \ll N_i)$
$(Idem)$	$\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D} \wedge (X \ll M) \wedge \mathscr{E}$	$\to$	$\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D} \wedge \mathscr{E}$
$(ToSubst)$	$N[\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D}]$	$\to$	$(N[X = M])[\mathscr{C} \wedge \mathscr{D}]$ if $X \notin \mathscr{D}om(\mathscr{C} \wedge \mathscr{D})$
$(Identity)$	$M[\mathtt{id}]$	$\to$	M
$(Replace)$	$X[\phi \wedge (X = M) \wedge \psi]$	$\to$	M
(Var)	$Y[\phi]$	$\to$	Y $\quad Y \notin \mathscr{D}om(\phi)$
$(Const)$	$c[\phi]$	$\to$	c
(Abs)	$(P \multimap M)[\phi]$	$\to$	$P[\phi] \multimap M[\phi]$
(App)	$(M\, N)[\phi]$	$\to$	$M[\phi]\, N[\phi]$
$(Struct)$	$(M \wr N)[\phi]$	$\to$	$M[\phi] \wr N[\phi]$
$(Constraint)$	$(M[P_i \ll N_i]_{i=1}^n)[\phi]$	$\to$	$(M[\phi])[P_i[\phi] \ll N_i[\phi]]_{i=1}^n$ $\quad n > 0$
$(Compose)$	$(N[X_i = M_i]_{i=1}^n)[\phi]$	$\to$	$N[\phi \wedge X_i = M_i[\phi]]_{i=1}^n$ $\quad n > 0$

Fig. 6 Small-step operational semantics of ρ_x°

4.2 Operational semantics of ρ_x°

The evaluation rules of ρ_x° are presented in Figure 6 and consist of those used for $\rho_\mathtt{s}$ and $\rho_\mathtt{m}$ together with a composition rule.

The application of rule abstractions and structures as well as the matching constraints decomposition are inherited from $\rho_\mathtt{m}$. As in $\rho_\mathtt{m}$, when part of the constraint is solved and independent of the rest of the constraint, the corresponding substitution should be applied. In ρ_x° the application of the substitution is just triggered (as in $\rho_\mathtt{s}$) in the rule $(ToSubst)$ and the rules inherited from $\rho_\mathtt{s}$ perform its application. The $(Identity)$ rule represents in fact two rules, one for the identity substitution and a second one for the identity constraint. When not clear from the context, the former is called $(Identity_s)$ while the latter is called $(Identity_c)$.

The newly introduced rule $(Constraint)$ distributes the substitutions in the constraints. The rule $(Compose)$ defines the composition of substitutions. The side condition for these two rules says that the constraint and respectively the substitution cannot be identities. For simplicity, we used an abuse of notation in the rule $(Compose)$ where $N[\phi \wedge X_i = M_i[\phi]]_{i=1}^n$ denotes the term $N[\phi \wedge X_1 = M_1[\phi] \wedge \ldots \wedge X_n = M_n[\phi]]$.

Example 10 (Multiple applications)
The composition of substitutions leads to more efficient evaluations. The following evaluation is obtained for the term considered in Example 5:

$$
\begin{array}{ll}
 & (g(X) \twoheadrightarrow ((f(Y) \twoheadrightarrow h(X,Y))\ f(a)))\ g(b) \\
\mapsto\!\!\!\!\mapsto & ((f(Y) \twoheadrightarrow h(X,Y))\ f(a))\,[X=b] \\
\mapsto\!\!\!\!\mapsto & h(X,Y)\,[Y=a]\,[X=b] \\
\mapsto_{Compose} & h(X,Y)\,[X=b \wedge Y=a\,[X=b]] \\
\mapsto_{Const} & h(X,Y)\,[X=b \wedge Y=a] \\
\mapsto\!\!\!\!\mapsto_{App} & h(X\,[X=b \wedge Y=a], Y\,[X=b \wedge Y=a]) \\
\mapsto\!\!\!\!\mapsto_{Replace} & h(b,a)
\end{array}
$$

All the evaluation steps dealing with the traversal of the term $h(X,Y)$ are done only once this time since only one substitution should be propagated.

The non-efficient route given in Example 5 can also be taken but while this was the only alternative for $\rho_{\mathbf{s}}$, this problem disappears for ρ°_{x} if we apply the evaluation rules with a strategy [6] that gives the highest priority to the composition rule, a canonical way to limit term traversal.

As mentioned at the beginning of this section, a ρ-calculus handling explicitly the constraint solving and the substitution application was coined for the first time in [12]. In ρ_{x} there was no mechanism for the composition of substitutions (*i.e.* no $(Compose)$ rule) and consequently, no substitution conjunctions. Therefore, we can consider ρ_{x} as a restriction of ρ°_{x} where all substitutions are simple equations and the $(Compose)$ rule is not available.

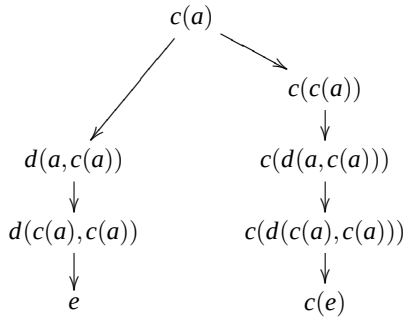
4.3 Properties of ρ°_{x}

The confluence of higher-order systems dealing with non-linear matching is still a difficult task since we usually obtain non-joinable critical pairs as those coined for the first time by Klop [35]. This counter example can be encoded in the ρ-calculus but the encoding is a bit tricky and is smoothly described in the following section. This presentation is a direct translation from [50].

4.3.1 Encoding Klop counter example in the ρ-calculus

Let us begin by describing Klop's counter example in classical rewriting. We recall that the non-confluence is due to a non-linear rewrite rule.

Example 11 [35] Let us consider the first-order rewrite system consisting of the rewrite rules $\{d(x,x) \to e, c(x) \to d(x,c(x)), a \to c(a)\}$. The following reductions are obtained when reducing the term $c(a)$:



We cannot close the diagram since e is in normal form and the smallest reduction from $c(e)$ to e would reduce $d(e,c(e))$ which can only be reduced by a reduction from $c(e)$ to e, which contradicts the minimality of the reduction.

We can go a step further in the encoding by considering λ-terms and higher-order rewriting. We introduce the following λ-terms and the corresponding reductions:

$$
\begin{array}{lll}
C & \equiv & \mathsf{Y}\big(\lambda y.\lambda x.d(x,yx)\big) \\
C & \mapsto\!\!\!\!\mapsto_{\beta} & \big(\lambda y.\lambda x.d(x,yx)\big)\,C \\
 & \mapsto_{\beta} & \lambda x.d(x,Cx)
\end{array}
\qquad\qquad
\begin{array}{lll}
A & \equiv & \mathsf{Y}C \\
A & \mapsto\!\!\!\!\mapsto_{\beta} & CA
\end{array}
$$

The λ-term C simulates the behavior of the constant c and of the second rewrite rule, whereas A simulates the behavior of the constant a and of the third rewrite rule. We can thus omit these two rules and consider a variation of Klop's example.

Example 12 If we add in the λ-calculus the non-linear rewrite rule $\mathscr{R} = \{d(x,x) \to e\}$ to the β-reduction then we obtain a non confluent calculus. In fact, the λ-term A reduces by $\mapsto_{\beta\cup\mathscr{R}}$, on the one hand to e and on the other hand to (Ce) and these two terms do not share a common reduct.

We are now ready to give the encoding in the ρ-calculus.

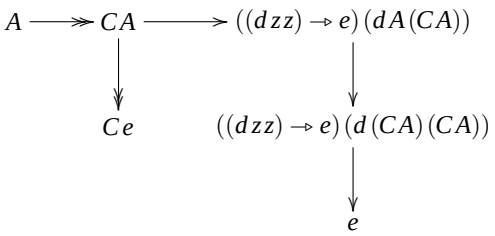
Example 13 The usual fixpoint combinator Y is defined as in the λ-calculus by:

$$\mathsf{Y} \triangleq \Big(y \twoheadrightarrow x \twoheadrightarrow (x(yyx))\Big)\Big(y \twoheadrightarrow x \twoheadrightarrow (x(yyx))\Big)$$

and for any ρ-term A we have $\mathsf{Y}A \mapsto\!\!\!\!\to_\rho A(\mathsf{Y}A)$. We now define the following terms as in the previous example except that the rule $d(z,z) \to e$ is directly encoded in C.

$$\begin{aligned} C &\equiv \mathsf{Y}\big(y \twoheadrightarrow x \twoheadrightarrow ((dzz) \twoheadrightarrow e)(dx(yx))\big) \\ C &\mapsto\!\!\!\!\to_\rho \big(y \twoheadrightarrow x \twoheadrightarrow ((dzz) \twoheadrightarrow e)(dx(yx))\big)\, C \\ &\mapsto\!\!\!\!\to_\rho x \twoheadrightarrow ((dzz) \twoheadrightarrow e)(dx(Cx)) \\ A &\equiv \mathsf{Y}C \end{aligned}$$

We have the following reductions:



The constant e is in normal form and the smallest reduction from (Ce) to e we must reduce the head redex. After several reductions, we obtain $((dzz) \twoheadrightarrow e)(de(Ce))$ which can be head reduced only after a reduction from (Ce) to e. Since the reduction is supposed to be minimal, we conclude by contradiction that (Ce) cannot be reduced to e and thus that the two reductions cannot be joined.

4.3.2 Proof scheme

As mentioned before, all the evaluation steps are performed modulo the theory of the conjunction (in ρ°_x modulo the associativity and the neutral elements) and thus, we are performing rewriting modulo a set of axioms. We give first a formal definition for this evaluation and then we introduce a more operational evaluation that is usually used in proofs and implementations. For a detailed exposition about rewriting modulo, we refer to [29, 30, 32, 44].

Definition 3
Given a rewrite system R and a set of axioms A, the term t (R/A)-rewrites to t', denoted $t \to_{R/A} t'$, if there exists a rule $l \to r \in R$, a term u, an occurrence ω in u and a substitution σ such that $t \overset{*}{\longleftrightarrow}_A u[\omega \hookleftarrow \sigma(l)]$ and $t' \overset{*}{\longleftrightarrow}_A u[\omega \hookleftarrow \sigma(r)]$ where $u[\omega \hookleftarrow v]$ denotes the term u with the sub-term at the position ω replaced by v.

Definition 4
Given a rewrite system R and a set of axioms A , a term t (R,A)-rewrites to a term t', which is denoted by $t \to_{R,A} t'$ if there exists a rule $l \to r \in R$, a position ω in t, and a substitution σ such that $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$ (*i.e.* the sub-term of t at position ω is equivalent to $\sigma(l)$), and $t' = t[\omega \hookleftarrow \sigma(r)]$ (*i.e.* t' is equal to t where the sub-term at the position ω is replaced by $\sigma(r)$).

In the following, we will denote by $A1$ the set of axioms defining the associativity and the neutral elements id_m and id_s for the conjunction and by $\sim_{A1}$ the equivalence relation induced by these axioms. In order to simplify the reading, we denote by $\mapsto_K$ the relation $\mapsto_{k,A1}$ where k is the rewrite relation induced by all the evaluation rules except (ρ) and (δ). Similarly, we denote by $\mapsto_\sigma$ the relation induced by the rules dealing with the application of substitutions (the $(Identity)$ rule for substitutions and the rules from $(Replace)$ to $(Compose)$). We denote ρ°_x the rewrite system given in Figure 6, and thus we obtain the relations $\mapsto_{\rho^\circ_x,A1}$ and $\mapsto_{\rho^\circ_x/A1}$.

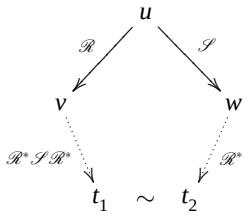
Definition 5 (Linear ρ°_x)
A pattern is *linear* if it does not contain two occurrences of the same variable. We say that a substitution $(X_i = M_i)_{i=1}^n, n > 0$ is *linear* if all the variables X_i are different. We say that a matching constraint $(P_i \ll M_i)_{i=1}^n, n > 0$ is *linear* if $\bigcap_{i=1}^n \mathscr{FV}(P_i) = \emptyset$. id_s and id_m are both linear.

The *linear* ρ°_x is the ρ°_x where all the patterns, substitutions and constraints are linear.

The general scheme of the proof of the confluence of the linear ρ°_x follows the proof of the confluence of the full theory of the $\lambda\sigma_{\Uparrow}$-calculus presented in [18] and uses a variant of Yokouchi-Hikita's lemma [51]:

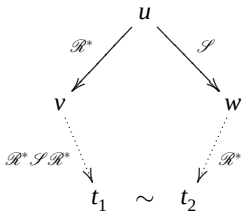
Lemma 1 *Let $\mathscr{R}$ and $\mathscr{S}$ be two relations defined on the same set $\mathscr{T}$ and $\sim$ an equivalence relation such that*

- *$\mathscr{R}$ is strongly normalizing.*
- *$\mathscr{R}$ is confluent modulo $\sim$, i.e., for all u,v,w in $\mathscr{T}$ such that $u\ \mathscr{R}^*\ v$ and $u\ \mathscr{R}^*\ w$ there exist t_1,t_2 in $\mathscr{T}$ such that $v\ \mathscr{R}^*\ t_1$, $w\ \mathscr{R}^*\ t_2$ and $t_1 \sim t_2$*
- *$\mathscr{S}$ has the diamond property, i.e., for all u,v,w in $\mathscr{T}$ such that $u\mathscr{S}v$ and $u\mathscr{S}w$ there exists an element t in $\mathscr{T}$ such that $v\mathscr{S}t$ and $w\mathscr{S}t$*
- *$\mathscr{R}$ and $\mathscr{S}$ are coherent modulo $\sim$, i.e., for all u,v,w such that $u \sim v$, $u\mathscr{R}w$ (resp. $u\mathscr{S}w$) there exists a t such that $v\mathscr{R}t$ (resp. $v\mathscr{S}t$) and $w \sim t$.*
- *the following diagram (often mentioned as Yokouchi-Hikita's diagram) holds:*

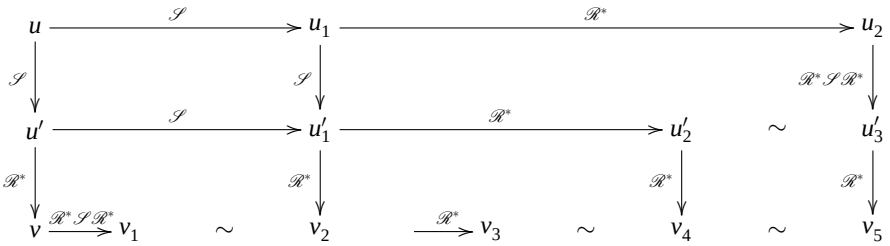


Then the relation $\mathscr{R}^*\mathscr{S}\mathscr{R}^*$ is confluent modulo $\sim$.

Proof Since the coherence properties are strong, to go from the classical lemma given in [18] to this generalization is just a straightforward verification. We only have to use these coherence properties as much as needed to "factorize" the coherence relation at the bottom of all commutative diagrams given in [18]. In fact, we first prove by induction on the $\mathscr{R}$-depth that:



Then we conclude by induction on the $\mathscr{R}$-depth of u, and where we distinguish between the depth zero and nonzero. The following diagram holds for the case zero



and the confluence follows since $\mathscr{R}$ is coherent with $\sim$ and thus $v_1 \sim v_2\ \mathscr{R}^*\ v_3$ can be replaced by $v_1\ \mathscr{R}^*\ v_3' \sim v_3$.

The diagram for the induction case is more complex but is handled similarly. □

We thus split the evaluation rules of ρ°_{x} into two relations corresponding to the relations $\mathscr{R}$ and $\mathscr{S}$ of the previous lemma. A natural choice for the relation $\mathscr{R}$ is $\mapsto_K$. In fact, the rules dealing with constraints and explicit substitutions should be strongly normalizing and confluent (in explicit substitution calculi we always ask these properties for the explicit part). The relation $\mathscr{S}$ cannot be the rewrite relation induced by the (ρ) and (δ) rules since this relation verifies neither the diamond property (the (δ) rule duplicates redexes) nor the Yokouchi-Hikita's diagram (the relation $\mapsto_K$ duplicates also redexes). This is why we use a parallelization of the (ρ), (δ) rules.

We first prove in Section 4.3.4 the termination of $\mapsto_K$. Then we prove that $\mapsto_K$ is confluent, taking into account the fact that rules can be applied modulo the associativity and the neutral elements (id_s, id_m) for the conjunctions. Then we formally define the parallelization of (ρ) and (σ) and show that it verifies the diamond property (Section 4.3.6). Finally, we prove the Yokouchi-Hikita's diagram and conclude the proof of the confluence of the linear ρ°_{x} by noticing that the parallelization and the original relation have the same transitive closure.

First of all, we show the soundness of the explicit substitution reductions, a mandatory and useful lemma in any calculus involving explicit substitution.

4.3.3 Soundness of explicit substitutions

First, we show that $\mapsto_\sigma$ is confluent and strongly normalizing and thus defines a function σ on terms. Secondly, we show that explicit substitutions soundly relate to meta substitutions. The corollary is that the function σ associates to every term a pure term (where all the substitutions have been applied).

Lemma 2 (Convergence of $\mapsto_\sigma$)
In the linear ρ°_{x}, the relation $\mapsto_\sigma$ is confluent and strongly normalizing.

Proof The relation $\mapsto_\sigma$ is strongly normalizing by Lemma 8 which proves a more general result that is, the termination of $\mapsto_K$. The local confluence is easily proved by observing that all critical pairs modulo A1 are convergent and as a consequence of the above properties, the confluence is obtained. We must notice that the linearity condition is mandatory since for example the non-linear substitution application $X\,[X = a \wedge X = b]$ leads either to a or to b. □

Lemma 3 (Soundness of explicit substitutions)
For all n, for all terms M_i and N, we have

$$N\,[X_i = M_i]_{i=1}^{n} \;\; \mapsto\!\!\!\!\!\rightarrow_{\mathcal{C}} \;\; N\{M_i/X_i\}_{i=1}^{n}$$

Proof If $n = 0$ then the result follows by the application of the $(Identity)$ rule. Otherwise, we proceed by induction on N.

- If N is a variable, then if this variable is one of the X_i for i between 1 and n then apply the $(Replace)$ rule. If not, then apply the (Var) rule.
- If N is an application, then $N_1\,N_2\,[X_i = M_i]_{i=1}^{n}$ can be reduced using the (App) rule to $N_1\,[X_i = M_i]_{i=1}^{n}\;N_2\,[X_i = M_i]_{i=1}^{n}$. Then, the result follows by induction.
- If N is a substitution application, then

$$\begin{array}{lll} N'[X_i = M_i]_{i=1}^{n}\,[\phi] & \mapsto_{Compose} & N'[\phi \wedge X_i = M_i\,[\phi]]_{i=1}^{n} \\ & \equiv_{(1)} & N'\{\phi, M_i\{\phi\}/X_i\}_{i=1}^{n} \\ & \equiv_{(2)} & N'\{\phi\}\{M_i\{\phi\}/X_i\}_{i=1}^{n} \\ & \equiv_{(3)} & N'\{M_i/X_i\}_{i=1}^{n}\{\phi\} \end{array}$$

  The first step (1) is the application of the induction hypothesis twice. The second step is valid since by α-conversion we can suppose that all the X_i do not belong to the free variables of M_i. The third step is exactly the substitution lemma of the plain ρ-calculus ($X_i \notin \mathscr{FV}(M_i)$).
- The other cases are similar to the application one.

□

4.3.4 Termination of the constraint handling rules

First of all, we will show that $\mapsto_K$ is strongly normalizing and for this we use the lexicographic product of two orders. The first order is a measure (a size) on terms such that the size of the right-hand side is smaller than that of the left-hand side for the rules $(ToSubst)$, $(Idem)$, $(Replace)$, and the decomposition rules and equal for all the other rules. The second order is based on a polynomial interpretation which decreases on all the other rules.

The measure on terms defined below represents (an upper bound of) the size of the corresponding pure term where all the pending substitutions were applied. The size of a term $N\,[X = M]$ is thus the size of N plus the size of M multiplied by the number of occurrences of X is N (called here the multiplicity of X in N), taking thus into account the possible instantiation(s) of X by M in N. As far as it concerns the matching constraints, we have to take into account that they can (possibly) become substitutions. We make thus an approximation and consider that for the terms of the form $N\,[P \ll M]$, each variable of P is (potentially) instantiated by a sub-term of M that is approximated by M.

This measure is preserved by the duplicative rules (*e.g.* the term $(X \wr X)\,[\phi]$ and its (*Struct*) reduct $X\,[\phi] \wr X\,[\phi]$ have the same size independently of ϕ). Notice also that in the right-hand side of the rule (*ToSubst*), M is not affected by the variables of the domain of $\mathscr{C} \wedge \mathscr{D}$ (by α-conversion) and thus, the size decreases (see Lemma 6).

Definition 6 (Multiplicity)
The multiplicity of the variable X in the term M, denoted $\mathfrak{M}_X(M)$, is defined inductively by:

$$\begin{array}{lll}
\mathfrak{M}_X(X) & = 1 & \mathfrak{M}_X(Y) = 0 \quad \text{if } X \neq Y \\
\mathfrak{M}_X(c) & = 0 & \mathfrak{M}_X(P \twoheadrightarrow M) = \mathfrak{M}_X(M) \\
\mathfrak{M}_X(M\,N) & = \mathfrak{M}_X(M) + \mathfrak{M}_X(N) & \mathfrak{M}_X(M \wr N) = \mathfrak{M}_X(M) + \mathfrak{M}_X(N) \\
\mathfrak{M}_X(N\,[\mathscr{C}]) & = \mathfrak{M}^{\mathfrak{c}}_X(N\,[\mathscr{C}]) + \mathfrak{M}_X(N) & \mathfrak{M}_X(N\,[\phi]) = \mathfrak{M}^{\mathfrak{c}}_X(N\,[\phi]) + \mathfrak{M}_X(N)
\end{array}$$

$$\begin{array}{lll}
\mathfrak{M}^{\mathfrak{c}}_X(N\,[\mathscr{C} \wedge \mathscr{D}]) & = \mathfrak{M}^{\mathfrak{c}}_X(N\,[\mathscr{C}]) + \mathfrak{M}^{\mathfrak{c}}_X(N\,[\mathscr{D}]) & \mathfrak{M}^{\mathfrak{c}}_X(N\,[\phi \wedge \psi]) = \mathfrak{M}^{\mathfrak{c}}_X(N\,[\phi]) + \mathfrak{M}^{\mathfrak{c}}_X(N\,[\psi]) \\
\mathfrak{M}^{\mathfrak{c}}_X(M\,[Y = N]) & = \mathfrak{M}_X(N) \times \mathfrak{M}_Y(M) & \mathfrak{M}^{\mathfrak{c}}_X(N\,[\mathtt{id}]) = 0
\end{array}$$

$$\mathfrak{M}^{\mathfrak{c}}_X(M\,[P \ll N]) = \sum_{Y_j \in \mathscr{FV}(P)} \mathfrak{M}_X(N) \left(\mathfrak{M}_{Y_j}(M) + 1 \right)$$

where $X \neq Y$ and $X \notin \mathscr{FV}(P)$

Definition 7 (Size)
The size of a term M, denoted $\mathfrak{S}(M)$, is defined as:

$$\begin{array}{lll}
\mathfrak{S}(X) & = 1 & \\
\mathfrak{S}(c) & = 1 & \mathfrak{S}(P \twoheadrightarrow M) = \mathfrak{S}(P) + \mathfrak{S}(M) \\
\mathfrak{S}(M\,N) & = \mathfrak{S}(M) + \mathfrak{S}(N) & \mathfrak{S}(M \wr N) = \mathfrak{S}(M) + \mathfrak{S}(N) \\
\mathfrak{S}(N\,[\mathscr{C}]) & = \mathfrak{S}^{\mathfrak{c}}(N\,[\mathscr{C}]) + \mathfrak{S}(N) & \mathfrak{S}(N\,[\phi]) = \mathfrak{S}^{\mathfrak{c}}(N\,[\phi]) + \mathfrak{S}(N)
\end{array}$$

$$\begin{array}{lll}
\mathfrak{S}^{\mathfrak{c}}(N\,[\mathscr{C} \wedge \mathscr{D}]) & = \mathfrak{S}^{\mathfrak{c}}(N\,[\mathscr{C}]) + \mathfrak{S}^{\mathfrak{c}}(N\,[\mathscr{D}]) & \mathfrak{S}^{\mathfrak{c}}(N\,[\phi \wedge \psi]) = \mathfrak{S}^{\mathfrak{c}}(N\,[\phi]) + \mathfrak{S}^{\mathfrak{c}}(N\,[\psi]) \\
\mathfrak{S}^{\mathfrak{c}}(M\,[Y = N]) & = \mathfrak{S}(N) \times \mathfrak{M}_Y(M) & \mathfrak{S}^{\mathfrak{c}}(N\,[\mathtt{id}]) = 0
\end{array}$$

$$\mathfrak{S}^{\mathfrak{c}}(M\,[P \ll N]) = \sum_{Y_j \in \mathscr{FV}(P)} \mathfrak{S}(N) \times \left(\mathfrak{M}_{Y_j}(M) + 1 \right)$$

We should point out that the notions of multiplicity and size are compatible *w.r.t.* to neutrality of the conjunction operator since `id` has no impact on the two notions. It is also compatible *w.r.t.* the associativity of the conjunction and *w.r.t.* α-conversion.

Lemma 4 (Soundness of multiplicity)
For any term M and variable X such that $X \notin \mathscr{FV}(M)$ *we have* $\mathfrak{M}_X(M) = 0$.

Proof By induction on the structure of M.

- $M = Y$ with $X \neq Y$. By definition.
- $M = c$. By definition.
- $M = P \twoheadrightarrow M_1$. We have $\mathfrak{M}_X(P \twoheadrightarrow M_1) = \mathfrak{M}_X(M_1)$ and the result holds by applying the induction hypothesis.
- $M = M_1\,M_2$. Then $\mathfrak{M}_X(M_1\,M_2) = \mathfrak{M}_X(M_1) + \mathfrak{M}_X(M_2)$ and the result holds by applying twice the induction hypothesis since $X \notin \mathscr{FV}(M_1M_2)$ implies $X \notin \mathscr{FV}(M_1)$, $X \notin \mathscr{FV}(M_2)$.
- $M = M_1 \wr M_2$. Similar to the previous case.
- $M = M_1\,[\mathscr{C}]$. We can apply the induction hypothesis to M_1 and get $\mathfrak{M}_X(M_1) = 0$. Then, we show (by structural induction on $\mathscr{C}$) that for all constraint $\mathscr{C}$, we have $\mathfrak{M}^{\mathfrak{c}}_X(M_1\,[\mathscr{C}]) = 0$
  - $\mathscr{C} = (P \ll N)$. Apply the induction hypothesis on N (induction on the set of terms).
  - $\mathscr{C} = (\mathscr{D}_1 \wedge \mathscr{D}_2)$ then the result follows by applying the induction hypothesis to $\mathscr{D}_1$ and $\mathscr{D}_2$ (induction on the set of constraint).

  - $\mathscr{C} = \mathtt{id}$. The result holds by definition.
- $M = M_1[\phi]$. We proceed as before and we use an induction on ϕ
  - $\phi = (Y = N)$. Apply the induction hypothesis to N.
  - $\phi = \psi_1 \wedge \psi_2$. The result follows by applying the induction hypothesis to ψ_1 and ψ_2 (induction on the set of substitutions).

□

Lemma 5 (Preservation of multiplicity)
For any variable X and any terms M and N such that $M \mapsto_K N$, the following inequality holds:

$$\mathfrak{M}_X(M) \geq \mathfrak{M}_X(N)$$

Proof We can prove that for all constraints $\mathscr{C}$, terms M and N such that $\mathscr{D}om(\mathscr{C}) \cap \mathscr{FV}(M) = \emptyset$ we have

$$\mathfrak{M}^c_X(N[Y = M][\mathscr{C}]) = \mathfrak{M}^c_X(N[\mathscr{C}]) \tag{1}$$

The proof is done by induction on $\mathscr{C}$. We only show the basic case, *i.e.*, if $\mathscr{C} = P \ll M'$.

$$\begin{aligned}
\mathfrak{M}^c_X(N[Y = M][P \ll M']) &= \sum_{Z_j \in \mathscr{FV}(P)} \mathfrak{M}_X(M') \left(\mathfrak{M}_{Z_j}(N[Y = M]) + 1\right) \\
&= \sum_{Z_j \in \mathscr{FV}(P)} \mathfrak{M}_X(M') \left(\mathfrak{M}_{Z_j}(M)\mathfrak{M}_Y(N) + \mathfrak{M}_{Z_j}(N) + 1\right) \\
&= \sum_{Z_j \in \mathscr{FV}(P)} \mathfrak{M}_X(M') \left(\mathfrak{M}_{Z_j}(M)\mathfrak{M}_Y(N) + \mathfrak{M}_{Z_j}(N) + 1\right) \\
&= \sum_{Z_j \in \mathscr{FV}(P)} \mathfrak{M}_X(M') \left(\mathfrak{M}_{Z_j}(N) + 1\right) \quad \text{By hypothesis and Lemma 4} \\
&= \mathfrak{M}^c_X(N[P \ll M'])
\end{aligned}$$

One can notice that the condition $X \notin \mathscr{FV}(M)$ is not needed since the multiplicities in M only depend on the variables in the domain of $\mathscr{C}$ (and not on X).

We can also prove by an easy induction on n that if X does not belong to $\cup_{i=1}^n \mathscr{FV}(P_i)$ and if for all i, $X \neq Y_i$ then

$$\mathfrak{M}_X\left(N\left[\bigwedge_{i=1..n} P_i \ll M_i\right]\right) = \mathfrak{M}_X(N) + \sum_{i=1}^n \sum_{X_j \in \mathscr{FV}(P_i)} \mathfrak{M}_X(M_i) \times \left(\mathfrak{M}_{X_j}(N) + 1\right) \tag{2}$$

$$\mathfrak{M}_X\left(N\left[\bigwedge_{i=1..n} Y_i = M_i\right]\right) = \mathfrak{M}_X(N) + \sum_{i=1}^n \mathfrak{M}_X(M_i) \times \mathfrak{M}_{X_i}(N) \tag{3}$$

Since the multiplicity is defined by a monotonic induction it is sufficient to prove the result when the reduction takes place at the root of M. We give all the computations for the relevant rules; they are similar for the remaining ones. In

particular, the case $(\mathit{Constraint})$ not given here uses the Eq. 2.

$$
\begin{array}{ll}
\text{(App)} & \mathfrak{M}_X\left((M\,N)\,[\phi]\right) \\
& = \mathfrak{M}^c_X\left((M\,N)\,[\phi]\right) + \mathfrak{M}_X\left(M\,N\right) \\
& = \mathfrak{M}^c_X\left(M\,[\phi]\right) + \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(M\,[\phi]\right) + \mathfrak{M}_X\left(N\right) \\
& = \mathfrak{M}_X\left(M\,[\phi]\;N\,[\phi]\right) \\
\text{(Replace)} & \mathfrak{M}_X\left(Y\,[\phi \wedge Y = M \wedge \psi]\right) \\
& = \mathfrak{M}_X\left(Y\right) + \mathfrak{M}^c_X\left(Y\,[\phi \wedge Y = M \wedge \psi]\right) \\
& = \mathfrak{M}_X\left(Y\right) + \mathfrak{M}^c_X\left(Y\,[\phi]\right) + \mathfrak{M}^c_X\left(Y\,[Y = M]\right) + \mathfrak{M}^c_X\left(Y\,[\psi]\right) \\
& \geq \mathfrak{M}^c_X\left(Y\,[Y = M]\right) \\
& = \mathfrak{M}^c_X\left(M\right) \times \mathfrak{M}_Y\left(Y\right) \\
& = \mathfrak{M}^c_X\left(M\right) \\
\text{(ToSubst)} & \mathfrak{M}_X\left(N\,[\mathscr{C} \wedge (Y \ll M) \wedge \mathscr{D}]\right) \\
& = \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{C} \wedge (Y \ll M) \wedge \mathscr{D}]\right) \\
& = \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{C}]\right) + \mathfrak{M}^c_X\left(N\,[Y \ll M]\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{D}]\right) \\
& = \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{C}]\right) + \mathfrak{M}_X\left(M\right) \times \left(\mathfrak{M}_Y\left(N\right) + 1\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{D}]\right) \\
& > \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{C}]\right) + \mathfrak{M}_X\left(M\right) \times \mathfrak{M}_Y\left(N\right) + \mathfrak{M}^c_X\left(N\,[\mathscr{D}]\right) \\
& \overset{Eq.\ 1}{=} \mathfrak{M}_X\left(N\right) + \mathfrak{M}^c_X\left(N\,[Y = M]\,[\mathscr{C}]\right) + \mathfrak{M}_X\left(M\right) \times \mathfrak{M}_Y\left(N\right) + \mathfrak{M}^c_X\left(N\,[Y = M]\,[\mathscr{D}]\right) \\
& = \mathfrak{M}_X\left(N\,[Y = M]\right) + \mathfrak{M}^c_X\left(N\,[Y = M]\,[\mathscr{C} \wedge \mathscr{D}]\right) \\
& = \mathfrak{M}_X\left(N\,[Y = M]\,[\mathscr{C} \wedge \mathscr{D}]\right) \\
\text{(Compose)} & \mathfrak{M}_X\left(N[X_j = M_j]_{j=1}^m [X_i = M_i]_{i=1}^n\right) \\
& \overset{Eq.\ 3}{=} \mathfrak{M}_X\left(N[X_j = M_j]_{j=1}^m\right) + \textstyle\sum_i \mathfrak{M}_X\left(M_i\right) \mathfrak{M}_{X_i}\left(N[X_j = M_j]_{j=1}^m\right) \\
& \overset{Eq.\ 3}{=} \mathfrak{M}_X\left(N\right) + \textstyle\sum_j \mathfrak{M}_X\left(M_j\right) \mathfrak{M}_{X_j}\left(N\right) + \sum_i \mathfrak{M}_X\left(M_i\right) \left(\mathfrak{M}_{X_i}\left(N\right) + \sum_j \mathfrak{M}_{X_i}\left(M_j\right) \mathfrak{M}_{X_j}\left(N\right)\right) \\
& = \mathfrak{M}_X\left(N\right) + \textstyle\sum_i \mathfrak{M}_X\left(M_i\right) \mathfrak{M}_{X_i}\left(N\right) + \sum_j \mathfrak{M}_{X_j}\left(N\right) \left(\mathfrak{M}_X\left(M_j\right) + \sum_i \mathfrak{M}_X\left(M_j\right) \mathfrak{M}_{X_i}\left(M_j\right)\right) \\
& = \mathfrak{M}_X\left(N\right) + \textstyle\sum_i \mathfrak{M}_X\left(M_i\right) \mathfrak{M}_{X_i}\left(N\right) + \sum_j \mathfrak{M}_X\left(M_j[X_i = M_i]_{i=1}^n\right) \\
& \overset{Eq.\ 3}{=} \mathfrak{M}_X\left(\left[X_i = M_i \wedge X_j = M_j\,[X_i = M_i]_{i=1}^n\right]_{i,j}\left(N\right)\right)
\end{array}
$$

□

Lemma 6 (Preservation of size)
For any terms M and N such that $M \mapsto_K N$, the following inequality holds:

$$\mathfrak{S}\left(M\right) \geq \mathfrak{S}\left(N\right)$$

The inequality is strict if the reduction is done using either the $(\mathit{ToSubst})$ rule, or the $(\mathit{Replace})$ rule, or the (Idem) rule, or the decomposition rules.

Proof The proof of the lemma is similar to that of Lemma 5. We only give some cases:

$$
\begin{array}{ll}
\text{(Replace)} & \mathfrak{S}\left(X\,[\phi \wedge X = M \wedge \psi]\right) \\
& = \mathfrak{S}\left(X\right) + \mathfrak{S}^c\left(X\,[\phi]\right) + \mathfrak{S}^c\left(X\,[X = M]\right) + \mathfrak{S}^c\left(X\,[\psi]\right) \\
& = 1 + \mathfrak{S}^c\left(X\,[\phi]\right) + \mathfrak{S}\left(M\right) \times \mathfrak{M}_X\left(X\right) + \mathfrak{S}^c\left(X\,[\psi]\right) \\
& > \mathfrak{S}\left(M\right) \\
\text{(ToSubst)} & \mathfrak{S}\left(X\,[\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D}]\right) \\
& = \mathfrak{S}\left(X\right) + \mathfrak{S}^c\left(X\,[\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D}]\right) \\
& = \mathfrak{S}\left(X\right) + \mathfrak{S}^c\left(X\,[\mathscr{C}]\right) + \mathfrak{S}^c\left(X\,[X \ll M]\right) + \mathfrak{S}^c\left(X\,[\mathscr{D}]\right) \\
& > 1 + \mathfrak{S}\left(M\right) \times \mathfrak{M}_X\left(X\right) \\
& > \mathfrak{S}\left(M\right) \\
\text{(Idem)} & \mathfrak{S}\left(N\,[\mathscr{C} \wedge (X \ll M) \wedge \mathscr{D} \wedge (X \ll M) \wedge \mathscr{E}]\right) \\
& = \mathfrak{S}\left(N\right) + \mathfrak{S}^c\left(N\,[\mathscr{C}]\right) + 2\mathfrak{S}^c\left(N\,[X \ll M]\right) + \mathfrak{S}^c\left(N\,[\mathscr{D}]\right) + \mathfrak{S}^c\left(N\,[\mathscr{E}]\right) \\
& > \mathfrak{S}\left(N\right) + \mathfrak{S}^c\left(N\,[\mathscr{C}]\right) + \mathfrak{S}^c\left(N\,[X \ll M]\right) + \mathfrak{S}^c\left(N\,[\mathscr{D}]\right) + \mathfrak{S}^c\left(N\,[\mathscr{E}]\right) \\
& = \mathfrak{S}\left(N\,[\mathscr{C} \wedge (X \ll M) \wedge \tau \wedge \mathscr{D}]\right)
\end{array}
$$

□

Definition 8 (Polynomial interpretation)
We use the standard order on natural numbers in order to define the following polynomial interpretation:

$$\begin{array}{lcllcl}
\mathfrak{I}(k) & = & 3 & \mathfrak{I}(P \twoheadrightarrow N) & = & \mathfrak{I}(P)+\mathfrak{I}(N)+1 \\
\mathfrak{I}(M \wr N) & = & \mathfrak{I}(M)+\mathfrak{I}(N)+1 & \mathfrak{I}(M\,N) & = & \mathfrak{I}(M)+\mathfrak{I}(N)+1 \\
\mathfrak{I}(N[\mathscr{C}]) & = & \mathfrak{I}(\mathscr{C})+\mathfrak{I}(N)+1 & \mathfrak{I}(M[\phi]) & = & (\mathfrak{I}(\phi)+2)\times\mathfrak{I}(M) \\
 & & & & & \\
\mathfrak{I}(\mathscr{C}\wedge\mathscr{D}) & = & \mathfrak{I}(\mathscr{C})+\mathfrak{I}(\mathscr{D}) & \mathfrak{I}(\phi\wedge\psi) & = & \mathfrak{I}(\phi)+\mathfrak{I}(\psi) \\
\mathfrak{I}(\mathtt{id}) & = & 0 & \mathfrak{I}(X=M) & = & \mathfrak{I}(M) \\
\mathfrak{I}(P \ll M) & = & \mathfrak{I}(P)+\mathfrak{I}(M) & & &
\end{array}$$

We should point out that the polynomial interpretation is compatible *w.r.t.* to neutrality of the conjunction operator since `id` has no impact on the two notions. It is also compatible *w.r.t.* the associativity of the conjunction and *w.r.t.* α-conversion. Moreover, since the addition and the multiplication are increasing on naturals, the monotonicity condition $a > b$ implies $\mathfrak{I}(a) > \mathfrak{I}(b)$ is clearly satisfied. We show that for any terms M and N such that $M \mapsto_K N$ the image of M is greater than that of N for any replacement of the interpretation of the variables of M and N by naturals bigger than 2.

Lemma 7 *For any terms M and N such that $M \mapsto_K N$ using either $(Compose)$, or $(Constraint)$, or (Abs), or $(Const)$, or (Var), or $(Identity)$, or (App), or $(Struct)$ then*

$$\mathfrak{I}(M) > \mathfrak{I}(N)$$

Proof For any natural number n, for any terms N, M_i for any patterns P_i and for any variables Y_i the following equalities hold:

$$\mathfrak{I}\left(N\left[\bigwedge_{i=1..n} P_i \ll M_i\right]\right) = \mathfrak{I}(N)+1+\sum_i(\mathfrak{I}(P_i)+\mathfrak{I}(M_i)) \tag{1}$$

$$\mathfrak{I}\left(N\left[\bigwedge_{i=1..n} Y_i = M_i\right]\right) = \mathfrak{I}(N)\times\left(2+\sum_i \mathfrak{I}(M_i)\right) \tag{2}$$

We can first remark that for any constraint $\mathscr{C}$ (resp. substitution ϕ) we have $\mathfrak{I}(\mathscr{C}) \geq 0$ (resp. $\mathfrak{I}(\phi) \geq 0$) and for any term M we have $\mathfrak{I}(M) > 2$.

($Identity_s$) $\mathfrak{I}(M[\mathtt{id}_s]) = (\mathfrak{I}(\mathtt{id}_s)+2)\times\mathfrak{I}(M) = 2\times\mathfrak{I}(M) > \mathfrak{I}(M)$

($Identity_c$) $\mathfrak{I}(M[\mathtt{id}_c]) = \mathfrak{I}(\mathtt{id}_c)+\mathfrak{I}(M)+1 = \mathfrak{I}(M)+1 > \mathfrak{I}(M)$

(Var) $\mathfrak{I}(Y[\phi]) = (\mathfrak{I}(\phi)+2)\times\mathfrak{I}(Y) > \mathfrak{I}(Y)$

(Const) $\mathfrak{I}(c[\phi]) = (\mathfrak{I}(\phi)+2)\times\mathfrak{I}(c) > \mathfrak{I}(c)$

(Abs) $\mathfrak{I}((P \twoheadrightarrow M)[\phi]) = (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(P \twoheadrightarrow M)$
$= (\mathfrak{I}(\phi)+2) \times (\mathfrak{I}(P)+\mathfrak{I}(M)+1)$
$> (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(P) + (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(M)+1$
$= \mathfrak{I}(P[\phi]) + \mathfrak{I}(M[\phi])+1$
$= \mathfrak{I}(P[\phi] \twoheadrightarrow M[\phi])$

(App) $\mathfrak{I}((M\,N)[\phi]) = (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(M\,N)$
$= (\mathfrak{I}(\phi)+2) \times (\mathfrak{I}(M)+\mathfrak{I}(N)+1)$
$> (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(M) + (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(N)+1$
$= \mathfrak{I}(M[\phi]) + \mathfrak{I}(N[\phi])+1$
$= \mathfrak{I}(M[\phi]\,N[\phi])$

(Struct) $\mathfrak{I}((M \wr N)[\phi]) = (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(M \wr N)+1$
$= (\mathfrak{I}(\phi)+2) \times (\mathfrak{I}(M)+\mathfrak{I}(N)+1)+1$
$> (\mathfrak{I}(\phi)+2) \times \mathfrak{I}(M)+1+(\mathfrak{I}(\phi)+2) \times \mathfrak{I}(N)+1$
$= \mathfrak{I}(M[\phi]) + \mathfrak{I}(N[\phi])+1$
$= \mathfrak{I}(M[\phi] \wr N[\phi])$

(Constraint) $\mathfrak{I}(M[P_i \ll N_i]_{i=1}^n[\phi]) \overset{Eq.\,1}{=} (\mathfrak{I}(\phi)+2) \times (\mathfrak{I}(M)+\sum_i \mathfrak{I}(P_i)+\sum_i \mathfrak{I}(N_i)+1)$

$> \sum_i((\mathfrak{I}(\phi)+2) \times \mathfrak{I}(P_i)) + \sum_i((\mathfrak{I}(\phi)+2) \times \mathfrak{I}(N_i)) + \mathfrak{I}(\phi) + \mathfrak{I}(M)+1$ □

$= \sum_i(\mathfrak{I}(P_i[\phi]) + \mathfrak{I}(N_i[\phi])) + \mathfrak{I}(\phi) + \mathfrak{I}(M)+1$

$= \sum_i \mathfrak{I}(P_i[\phi] \ll N_i[\phi]) + \mathfrak{I}(M[\phi])+1$

$= \mathfrak{I}(M[\phi][P_i[\phi] \ll N_i[\phi]]_{i=1}^n)$

(Compose) $\mathfrak{I}\left(N[X_i = M_i]_{i=1}^n[\phi]\right) = (\mathfrak{I}(\phi)+2) \times \mathfrak{I}\left(N[X_i = M_i]_{i=1}^n\right)$

$\overset{Eq.\,2}{=} (\mathfrak{I}(\phi)+2) \times (\sum_i \mathfrak{I}(M_i)+2) \times \mathfrak{I}(N)$

$= \mathfrak{I}(N) \times (2 \times (\mathfrak{I}(\phi)+2) + (\mathfrak{I}(\phi)+2) \times \sum_i \mathfrak{I}(M_i))$

$> \mathfrak{I}(N) \times (\mathfrak{I}(\phi)+2+(\mathfrak{I}(\phi)+2) \times \sum_i \mathfrak{I}(M_i))$

$= \mathfrak{I}(N) \times (\mathfrak{I}(\phi)+2+\sum_i \mathfrak{I}(M_i[\phi]))$

$= \mathfrak{I}(N)\,(\mathfrak{I}(\phi)+2+\sum_i \mathfrak{I}(X_i = M_i[\phi])) = \left(\mathfrak{I}\left(\phi \wedge \bigwedge_i (X_i = M_i[\phi])\right)+2\right) \times \mathfrak{I}(N)$

$= \mathfrak{I}\left(N[\phi \wedge X_i = M_i[\phi]]_{i=1}^n\right)$

Lemma 8
The relation $\mapsto_K$ is strongly normalizing.

Proof Every pair of terms in $\mapsto_K$ (strictly) decreases the lexicographic product $(\mathfrak{S}(), \mathfrak{I}())$. □

4.3.5 Confluence of the sub-relations

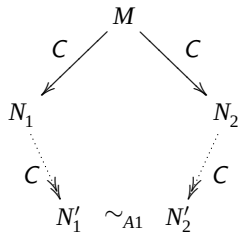
First, we show that $\mapsto_K$ is coherent modulo A1, *i.e.*, two equivalent (modulo A1) terms can be reduced to two equivalent ones.

Lemma 9 (Coherence modulo A1)
For any terms M, N, M' such that $M \mapsto_K N$ and $M \sim_{A1} M'$, there exists N' such that $N \sim_{A1} N'$ and $M' \mapsto_K N'$.

$$\begin{array}{ccc} M & \sim_{A1} & M' \\ {\scriptstyle C}\downarrow & & \vdots\,{\scriptstyle C} \\ N & \sim_{A1} & N' \end{array}$$

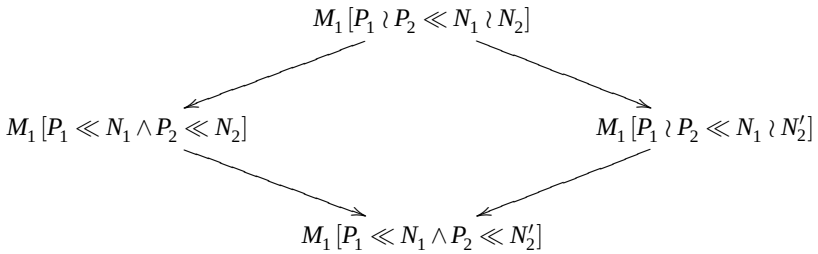
Proof The proof is done by case analysis on the rule used to reduced M into N. The diagram is easily closed for all rules except for the $(Idem)$ rule, for which we may observe that, thanks to the extension variables (the variables called $\mathscr{C}$ and $\mathscr{E}$ in the $(Idem)$ rule), all the computations from M to M' can be reproduced when performing the $\mapsto_K$ reductions from M' to N'. □

Lemma 10 (Local confluence modulo A1)
For any terms M, N_1, N_2 such that $M \mapsto_K N_1$ and that $M \mapsto_K N_2$, there exist two terms N_1' and N_2' such that $N_1 \mapsto\!\!\!\to_C N_1'$ and $N_2 \mapsto\!\!\!\to_C N_2'$ with $N_1' \sim_{A1} N_2'$:



Proof We proceed by induction on M. We suppose that the redexes are not disjoint (otherwise the result is easy to prove). In what follows, we call "the first reduction" the reduction from M to N_1 and "the second reduction" the reduction from M to N_2.

- If $M = M_1\, M_2$ then the two reductions take place in the same M_i. The result follows by applying the induction hypothesis.
- If $M = M_1[\mathscr{C}]$ then we proceed by induction on $\mathscr{C}$. If the two reductions take place in M_1 then the result follows by induction.
  - If $\mathscr{C} = id_m$ then the result is obvious.
  - If $\mathscr{C} = P \ll M_2$ then, if the two reductions take place in M_2 then the result follows by induction. If the first reduction is a decomposition rule, lets say $(Decompose_\wr)$ then



    If the first reduction is $(ToSubst)$ then just swap the two reduction steps.
  - If $\mathscr{C} = \mathscr{C}_1 \wedge \mathscr{C}_\in$ then, if the two reductions take place in $\mathscr{C}_1$ or in $\mathscr{C}_2$ then the result follows by induction. If the first reduction is $(ToSubst)$ then just swap the two reduction steps. Otherwise the first reduction reduces $\mathscr{C}$ at its top position and thus $\mathscr{C}$ must be equal modulo A1 to $\mathscr{D}_1 \wedge (X \ll N) \wedge \mathscr{D}_2 \wedge (X \ll N) \wedge \mathscr{D}_3$. If the second reduction takes place in one of the $\mathscr{D}_i$ then the result follows easily. If the second reduction occurs in one occurrence of N by reducing it to N' then close the diagram on the left by reducing the remaining occurrence of N into N' and on the right by first reducing the other occurrence of N into N' and then apply the $(Idem)$ rule. If the two reductions use the rule $(Idem)$ then there are two cases (any permutation of constraints in the first case is similar) :
    We can have
    $$\mathscr{C} \sim_{A1} \left(\mathscr{C}_1 \wedge (X \ll M) \wedge \mathscr{C}_2 \wedge (Y \ll N) \wedge \mathscr{C}_3 \wedge (X \ll M) \wedge \mathscr{C}_4 \wedge (Y \ll N) \wedge \mathscr{C}_5\right)$$
    and the first reduction eliminates the doubletons related to X and the second reduction eliminates the doubletons related to Y. Then, to close the diagram simply swap the two reductions (this is possible thanks to the extensions variables in the $(Idem)$ rule).
    Otherwise
    $$\mathscr{C} \sim_{A1} \left(\mathscr{C}_1 \wedge (X \ll M) \wedge \mathscr{C}_2 \wedge (X \ll M) \wedge \mathscr{C}_3 \wedge (X \ll M) \wedge \mathscr{C}_4\right)$$
    and conclude the case as in the previous case.
- If $M = M_1[\phi]$ then we proceed by induction on ϕ and by case analysis on the rule used for the first reduction.
  The interesting case is when the first reduction uses the $(Constraint)$ rule and the second reduction is done using the $(ToSubst)$ rule. In this case, the result follows using the $(Compose)$ rule and Lemma 3. Let us denote $\wedge_i(P_i \ll N_i)$ by $\mathscr{C}$ and $\wedge_j(P_j \ll N_j)$ by $\mathscr{D}$. In the following the simply write $\mathscr{C}[X = N]$ for $\wedge_i(P_i[X = N] \ll N_i[X = N])$. Let us suppose

moreover that $M_1 = M_3[\mathscr{C} \wedge (Y \ll M_2) \wedge \mathscr{D}]$ in which case the first reduction gives:

$$\begin{array}{ll} & (M_3[\mathscr{C} \wedge (Y \ll M_2) \wedge \mathscr{D}])[X = N] \\ \mapsto_{Constraint} & M_3[X = N][\mathscr{C}[X = N] \wedge (Y[X = N] \ll M_2[X = N]) \wedge \mathscr{D}[X = N]] \\ \mapsto_{Var} & M_3[X = N][\mathscr{C}[X = N] \wedge (Y \ll M_2[X = N]) \wedge \mathscr{D}[X = N]] \\ \mapsto_{ToSubst} & M_3[X = N][Y = M_2[X = N]][\mathscr{C}[X = N] \wedge \mathscr{D}[X = N]] \\ \mapsto_{Compose} & M_3[Y = M_2[X = N] \wedge X = N[Y = M_2[X = N]]][\mathscr{C}[X = N] \wedge \mathscr{D}[X = N]] \end{array}$$

By α-conversion, $Y \notin \mathscr{FV}(N)$ and so, by applying Lemma 3, we get

$$M_3[Y = M_2[X = N] \wedge X = N][\mathscr{C}[X = N] \wedge \mathscr{D}[X = N]] \qquad (1)$$

The second reduction gives:

$$\begin{array}{ll} & (M_3[\mathscr{C} \wedge (Y \ll M_2) \wedge \mathscr{D}])[X = N] \\ \mapsto_{ToSubst} & (M_3[Y = M_2][\mathscr{C} \wedge \mathscr{D}])[X = N] \\ \mapsto_{Constraint} & M_3[Y = M_2][X = N][\mathscr{C}[X = N] \wedge \mathscr{D}[X = N]] \\ \mapsto_{Compose} & M_3[X = N \wedge Y = M_2[X = N]][\mathscr{C}[X = N] \wedge \mathscr{D}[X = N]] \qquad (2) \end{array}$$

Lemma 3 concludes the case showing that the terms (1) and (2) are equal.

□

Lemma 11
The relation $\mapsto_K$ is confluent modulo A1.

Proof We actually prove a stronger property, that is that the relation is Church-Rosser modulo A1, which is obtained according to [44] from the strong normalization (Lemma 8), the coherence (Lemma 9) and the local confluence (Lemma 10) modulo A1 of $\mapsto_K$. □

4.3.6 Parallel version of the (ρ),(δ) rules

The parallelization of $\rho\delta$ intuitively reduces redexes in parallel. The definition can be easily deduced from the one of the λ-calculus (see for example [18]) as already done for the plain ρ-calculus [3].

Definition 9 (Parallelization of $\rho\delta$)
The parallelization of the relation induced by the rules (ρ) and (δ), denoted $\mapsto_{\rho\delta_\parallel}$, is defined inductively by

$$\frac{}{M \mapsto_{\rho\delta_\parallel} M} \qquad \frac{M \mapsto_{\rho\delta_\parallel} M'}{P \twoheadrightarrow M \mapsto_{\rho\delta_\parallel} P \twoheadrightarrow M'}$$

$$\frac{M \mapsto_{\rho\delta_\parallel} M' \quad N \mapsto_{\rho\delta_\parallel} N'}{M\,N \mapsto_{\rho\delta_\parallel} M'\,N'} \qquad \frac{M \mapsto_{\rho\delta_\parallel} M' \quad N \mapsto_{\rho\delta_\parallel} N'}{(P \twoheadrightarrow M)\,N \mapsto_{\rho\delta_\parallel} M'[P \ll N']}$$

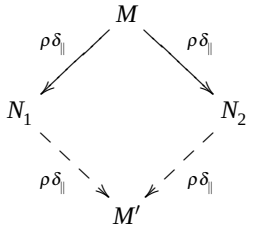
$$\frac{M_1' \mapsto_{\rho\delta_\parallel} M_1' \quad M_2 \mapsto_{\rho\delta_\parallel} M_2' \quad N \mapsto_{\rho\delta_\parallel} N'}{(M_1 \wr M_2)\,N \mapsto_{\rho\delta_\parallel} M_1'\,N' \wr M_2'\,N'} \qquad \frac{M \mapsto_{\rho\delta_\parallel} M' \quad N \mapsto_{\rho\delta_\parallel} N'}{M \wr N \mapsto_{\rho\delta_\parallel} M' \wr N'}$$

$$\frac{\mathscr{C} \mapsto_{\rho\delta_\parallel} \mathscr{C}' \quad M \mapsto_{\rho\delta_\parallel} M'}{M[\mathscr{C}] \mapsto_{\rho\delta_\parallel} M'[\mathscr{C}']} \qquad \frac{\phi \mapsto_{\rho\delta_\parallel} \phi' \quad M \mapsto_{\rho\delta_\parallel} M'}{M[\phi] \mapsto_{\rho\delta_\parallel} M'[\phi']}$$

$$\frac{N \mapsto_{\rho\delta_\parallel} N'}{(P \ll N) \mapsto_{\rho\delta_\parallel} (P \ll N')} \qquad \frac{N \mapsto_{\rho\delta_\parallel} N'}{(X = N) \mapsto_{\rho\delta_\parallel} (X = N')}$$

$$\frac{\mathscr{C} \mapsto_{\rho\delta_\parallel} \mathscr{C}' \quad \mathscr{D} \mapsto_{\rho\delta_\parallel} \mathscr{D}'}{\mathscr{C} \wedge \mathscr{D} \mapsto_{\rho\delta_\parallel} \mathscr{C}' \wedge \mathscr{D}'} \qquad \frac{\phi \mapsto_{\rho\delta_\parallel} \phi' \quad \psi \mapsto_{\rho\delta_\parallel} \psi'}{\phi \wedge \psi \mapsto_{\rho\delta_\parallel} \phi' \wedge \psi'}$$

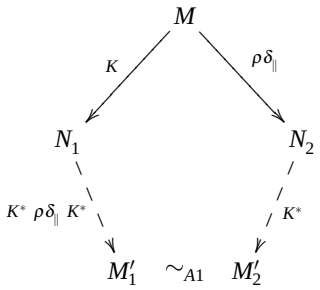
Lemma 12 (Diamond Property of $\mapsto_{\rho\delta_{\|}}$)
For any terms M, N_1, N_2 there exists a term M' such that the following diagram holds:



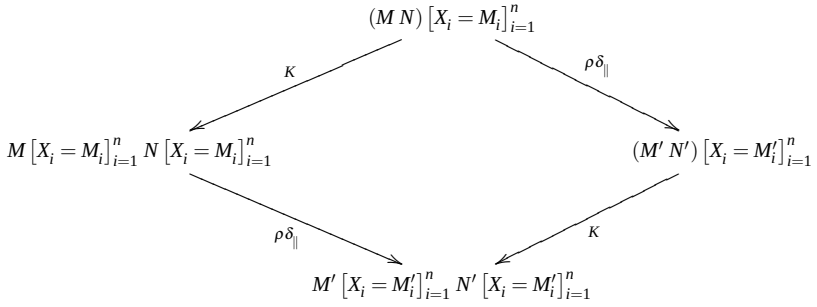
Proof The relation $\mapsto_{\rho\delta_{\|}}$ is the parallelization of an orthogonal system. □

4.3.7 Yokouchi-Hikita's diagram and the confluence of ρ°_{x}

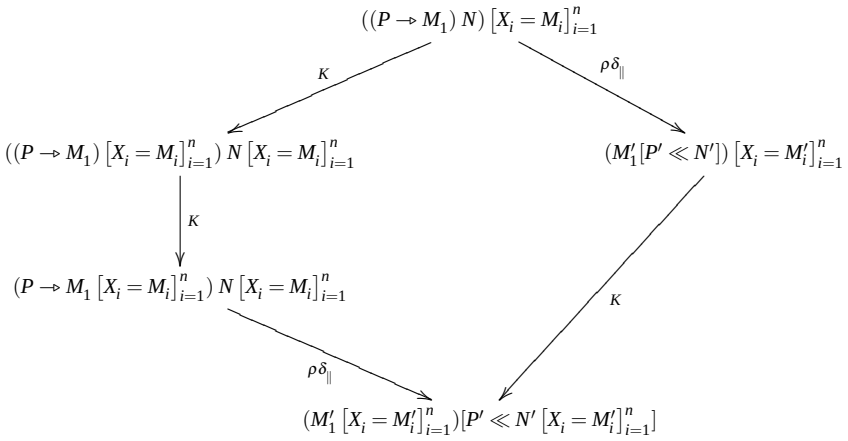
Lemma 13 (Yokouchi-Hikita's diagram)
For any terms M, N_1 and N_2 in the linear ρ°_{x} *there exists the terms M'_1, M'_2 such that the following diagram holds:*



Proof When the two steps from M to N_1 and from M to N_2 do not overlap, the lemma is easy. So we have to inspect every critical pair[1]. Since a strict subexpression of a $\rho\delta_{\|}$ redex can never overlap with a C redex, it is sufficient to work by cases on the derivation from M to N_1. We only consider the (App) rule. The other cases are simpler and similar.

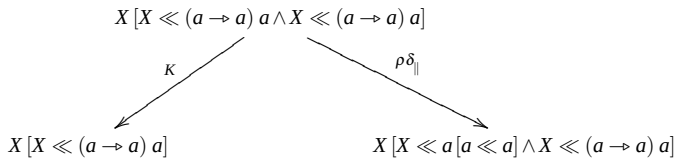


[1] As in the $\lambda\sigma_{\Uparrow}$-calculus a critical pair has a slightly different meaning than the standard one because of the parallel reduction.



The case where the $\rho\delta_{\|}$ reduction from M to N_2 concerns a δ redex is similar to the previous one.

Notice that the linearity condition is essential here since it ensures that the rule $(Idem)$ is never used. If this condition is not enforced and non-linear terms are allowed, the following (non-joinable) diagram gives a counterexample of Yokouchi-Hikita's diagram:



□

Lemma 14
The relation $\longmapsto\!\!\!\!\rightarrow_{\mathsf{C}} \mapsto_{\rho\delta_{\|}} \longmapsto\!\!\!\!\rightarrow_{\mathsf{C}}$ *is confluent modulo A1.*

Proof All the hypotheses of the Yokouchi-Hikita's lemma (Lemma 1) are proved in the previous lemmas:

- The relation $\mapsto_K$ is strongly normalizing by Lemma 8.
- The relation $\mapsto_K$ is confluent modulo A1 by Lemma 11.
- The relation $\mapsto_{\rho\delta_{\|}}$ has the diamond property by Lemma 12.
- The relations $\mapsto_K$ and $\mapsto_{\rho\delta_{\|}}$ are coherent modulo A1: the coherence of $\mapsto_{\rho\delta_{\|}}$ modulo A1 is obvious and the coherence of $\mapsto_K$ is obtained by Lemma 9.
- Yokouchi-Hikita's diagram holds by Lemma 13.

□

Lemma 15
The relation $\mapsto_{\rho^{\circ}_{\boldsymbol{x}},A1}$ *is confluent modulo A1.*

Proof We have $\mapsto_{\rho\delta} \subseteq \mapsto_{\rho\delta_{\|}} \subseteq \twoheadrightarrow_{\rho\delta}$ thus $\mapsto_{\rho^{\circ}_{\boldsymbol{x}},A1} \subseteq \longmapsto\!\!\!\!\rightarrow_{\mathsf{C}} \mapsto_{\rho\delta_{\|}} \longmapsto\!\!\!\!\rightarrow_{\mathsf{C}} \subseteq \twoheadrightarrow_{\rho^{\circ}_{\boldsymbol{x}},A1}$ and the reflexive and transitive closure of $(\longmapsto\!\!\!\!\rightarrow_{\mathsf{C}} \mapsto_{\rho\delta_{\|}} \longmapsto\!\!\!\!\rightarrow_{\mathsf{C}})$ is equal to $\twoheadrightarrow_{\rho^{\circ}_{\boldsymbol{x}},A1}$. Then the confluence modulo A1 of $\mapsto_{\rho^{\circ}_{\boldsymbol{x}},A1}$ is equivalent to the confluence modulo A1 of $\longmapsto\!\!\!\!\rightarrow_{\mathsf{C}} \mapsto_{\rho\delta_{\|}} \longmapsto\!\!\!\!\rightarrow_{\mathsf{C}}$. Lemma 14 concludes the proof. □

Theorem 1
The linear $\rho^{\circ}_{\boldsymbol{x}}$ *is confluent (the relation* $\mapsto_{\rho^{\circ}_{\boldsymbol{x}}/A1}$ *is confluent).*

Proof The property follows from the previous lemma and the coherence of $\mapsto_{\rho^{\circ}_{\boldsymbol{x}},A1}$ modulo $\sim_{A1}$ [44]. □

4.4 Possible extensions

In this paper, we only consider unitary matchings in the empty theory. In practice, it is interesting to have the possibility to reason modulo (equational) theories *w.r.t.* the defined constants. This can be done by adjusting the part of the calculus dealing with explicit matching. For example, the $(Decompose_{\mathscr{F}})$ rule can be adapted according to the theory one wants to deal with. For example, if we want to deal with commutative symbols like f_c (not necessarily binary) we obtain the rule $(Decompose^{C}_{\mathscr{F}})$ - where $\mathfrak{S}_n$ denotes the permutations of $\{1,\ldots,n\}$:

$$(Decompose^{C}_{\mathscr{F}}) \quad f_c(M_1,\ldots,M_n) \ll f_c(N_1,\ldots,N_n) \;\rightarrow\; \wr_{\varphi\in\mathfrak{S}_n}\left(\bigwedge_{i=1}^{n}(M_i \ll N_{\varphi(i)})\right)$$

If one prefers the AC (associative-commutative) matching theory, the corresponding decomposition rule should be specified.

Moreover, in most of the applications, the empty theory for the structure operator "$\wr$" is not sufficient. For example, if one wants to encode multisets, the AC theory should be used. If one wants to encode rewrite systems in the ρ-calculus, one needs a special theory for the structure so as to erase matching failures (this theory is presented in [16]). In such cases, the rule $Decompose_{\wr}$ should be modified in order to take into account the chosen theory and the conditions for the confluence of the obtained calculus are under investigation.

We can also point out that deciding in the empty theory whether a matching constraint has a solution is equivalent to solving it. In a more general context like, for example, associative and commutative matching, solving the matching problem can be significantly more complex then deciding whether a solution exists. A first step towards an efficient approach would be the use of "labeled" constraints that allow one to identify matching problems with at least a solution and matching problems with no solution. The rules dealing with constraints will be extended to label solvable parts of constraints. Then, the application of such a constraint labeled as solvable by an $\mathbf{s}$ (*e.g.* $\mathscr{E}^{\mathbf{s}}$) and independent of the remaining part, can be done in a more efficient way by using a modified rule $(ToSubst)$ and a new rule $(Independent)$:

$$\begin{array}{lllll}
(ToSubst) & N[\mathscr{C}\wedge\mathscr{E}^{\mathbf{s}}\wedge\mathscr{D}] & \rightarrow & (N[\mathscr{E}^{\mathbf{s}}])[\mathscr{C}\wedge\mathscr{D}] & \text{if } \mathscr{D}om(\mathscr{E}^{\mathbf{s}})\cap\mathscr{D}om(\mathscr{C}\wedge\mathscr{D})=\emptyset \\
(Independent) & N[\mathscr{C}^{\mathbf{s}}] & \rightarrow & N & \text{if } \mathscr{D}om(\mathscr{C}^{\mathbf{s}})\cap\mathscr{FV}(N)=\emptyset
\end{array}$$

In the $\rho_{\mathbf{s}}$ labels are implicit: "$X \ll M$" is labeled as a solvable constraint. The above two rules can also be used but, as we said before, the efficiency is not improved when considering a syntactic matching.

5 Implementation of explicit ρ-calculi

Explicit substitution calculi have been studied from different points of view. Different calculi have been proposed so as to obtain meta properties like confluence and preservation of strong normalization [37, 46, 18, 19]. They have been used to perform higher-order unification [22] just like to represent incomplete proofs in type theory [41] or to prove correctness of compiler optimizations [36]. Moreover, explicit substitution calculi have been used in two significant practical systems [42]: the Standard ML of New Jersey compiler and the Teyjus implementation of Lambda Prolog. [38] in particular precisely shows that the use of de Bruijn indices and the ability to merge together structure traversals (*i.e.* the composition of substitution) have a strong positive impact on the system performances.

The study of explicit ρ-calculi is thus also important for future implementations of languages based on the ρ-calculus. Actually, the ρ-calculus provides the basis for a language combining functional language and rewrite based language (ELAN, Maude...) features and thus besides providing a toy interpreter allowing us to experiment with the ρ-calculus, our implementation of explicit ρ-calculi is a first step toward such a language.

The rest of the section is devoted to the brief presentation of our implementation and of the support language, TOM. The gap between the different calculi defined in the previous sections and the actual implementation is rather small and so, we mainly focus here on the key features of TOM that led to a natural implementation of the explicit ρ-calculi presented in this paper.

Following the experience of ELAN, a strategic rewriting based language [8, 33], TOM was introduced to integrate rewriting into existing programming environments. A full presentation of TOM [2] is out of the scope of this paper but we refer to [40] for a detailed presentation.

[2] `http://tom.loria.fr`

1. TOM performs list matching (unlike ELAN that performs associative and commutative matching). Associative matching gives the opportunity to match against associative symbols whereas list matching allows one to match lists which, in general, have a different type universe. Constraints and substitutions are represented by lists of atomic matching equations (built using the operator "$\ll$") and respectively equations (built using the operator " $=$ ").
2. The overall evaluation process of a term with respect to a rewrite system and a given strategy can be implemented straightforwardly in TOM since one can define separately the rewrite system and the strategy to evaluate it. This is a good way to obtain a modular and easily maintainable implementation.
   Schematically, the application of a rewrite system $\{l_1 \twoheadrightarrow r_1, l_2 \twoheadrightarrow r_2, \ldots,\}$ guided by a strategy `eval` can be implemented, for example, by a function `apply` that has the following shape:
   ```
   % match(s){
      l1 -> return r1;
      l2 -> return r2;
      ...
   }
   return eval(s);
   ```
   With such a construction we try to apply one of the rules at the head position and if this is not possible (*i.e.* no l_i matches the subject s) apply the strategy `eval` describing how rules must be applied to sub-terms.
3. Thanks to an intensive use of the ATerm library [9], we obtain for free the corresponding maximal sharing representation of terms for a given signature. Maximal sharing has strong impact on performances. First, we can test the equality of terms in constant time (check the equality of memory addresses). Secondly, looping terms (like the classical λ-term $\omega\omega = (x \twoheadrightarrow xx)\ (x \twoheadrightarrow xx)$) loop forever and do not cause a stack overflow error (unlike in the implementation of the imperative ρ-calculus [3] [39]).

The implementation of the operational semantics of the ρ°_{x} follows the overall evaluation process given earlier. All previously given examples can be simulated by the implementation. Our interpreter is available on the ρ-calculus web page[3].

Conclusion and future work

We have proposed a ρ-calculus that handles explicitly the (syntactic) matching constraints and the application of the resulted substitutions. We have proved that it enjoys the classical properties of such a formalism, *i.e.*, the confluence of the calculus and the termination and confluence of the constraint handling part. We have seen that the calculus is modular and can be adapted to other matching theories for the defined constants and for the structure operator "$\wr$". We have shown that we can either choose to be atomic and give a simple definition of substitutions (as in $\rho_{\mathbf{s}}$), or more general and efficient and define a calculus that handles substitution composition (as in ρ°_{x}).

ρ-calculi and especially the ρ°_{x}, are new frameworks that provide us with theoretical foundations for a new family of programming languages. Different extensions/variations of the ρ-calculus are now available: in [39] an imperative version of the calculus has been proposed and in [24] exceptions in the ρ-calculus were studied. One can mention that the ρ-calculus allows one to design extremely powerful type systems such as those presented in [15,3,50]. The implementation briefly described in this paper can be seen as the basis for programming language incorporating the features introduced in these different extensions.

Related work: In [7], a calculus called the PSA-Calculus was introduced. The explicit application of a rewrite rule and the *explicit matching handling* were coined for the first time in this ancestor of the ρ-calculus. Nevertheless, it was a first approach to make explicit rewriting and thus less powerful than the current ρ-calculus. For example, the PSA-Calculus is not powerful enough to express strategies as explicit objects and thus there is a hierarchy between rules and strategies.

A rewriting calculus with explicit substitutions has been already proposed in [11]. This calculus is mainly an extension of the $\lambda\sigma$-calculus and is called the $\rho\sigma$-calculus. The approach is less general than the one presented here since this calculus makes explicit the substitution application but not the computations to go from constraints to substitutions. In [43], the cooperation between Coq and ELAN that automates the use of AC-rewriting is the proof assistant makes an intensive usage of the $\rho\sigma$-calculus to represent proof terms of rewrite derivations. The explicit treatment of matching in the ρ°_{x} should be a useful tool to obtain normalization traces in some non-trivial matching theories.

Our work follows the line of the works on explicit substitution calculi and more generally on the way to represent higher-order languages [21,4,5,45]. This paper shows that these works need to be extended to deal with the interaction of matching

[3] `http://rho.loria.fr/implementations.html`

and substitution. Even the works on generic calculi of explicit substitutions [48] cannot be used since, to the best of our knowledge, they do not handle composition of substitutions. Of course, if such works would be extended, the embedding of our calculus should be of interest.

Future work: Besides the extensions presented in Section 4.4 there are different points that should be studied

- understand how the approach proposed in [31] and, in particular, the permutation of substitutions can be applied in order to simplify the notion of substitution for the ρ-calculus.
- deal explicitly with α-conversion. There are many works that should be considered: deBruijn [20] indices, the $\lambda\lambda$-Calculus with explicit scoping [26], director strings [47].
- propose a powerful named exception mechanism that takes advantage of the very general management of errors. A first approach has been sketched in [15].
- extend explicit first-order syntactic matching first to equational matching and then to higher-order matching.

More generally, we want to understand the features that one wants to propose in programming language based on the ρ-calculus and how these features should be implemented. This question is strongly related to our intend to study integrated programming and proving environments where computations and deductions are uniformly integrated, *i.e.*, to unify functional and rewriting based languages (*e.g.*, ML, ELAN, Maude), proof assistants and theorem provers (*e.g.*, Coq, Isabelle, PVS, ...).

Acknowledgements

This work beneficited of the many discussions we had with Clara Bertolissi, Luigi Liquori, Benjamin Wack and of the constructive and useful comments of the anonymous referees.

References

1. Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, October 1991.
2. Henk Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.
3. Gilles Barthe, Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Pure patterns type systems. In *Principles of Programming Languages - POPL2003, New Orleans, USA*. ACM, January 2003.
4. Klaus Berkling and Elfriede Fehr. A consistent extension of the lambda-calculus as a base for functional programming languages. *Information and Control*, 55(1-3):89–101, October/November/December 1982.
5. Roel Bloo and Kristoffer Høgsbro Rose. Combinatory reduction systems with explicit substitution that preserve strong nomalisation. In *Proceedings of the Fifth International Conference on Rewriting Techniques and Application (RTA '96)*, pages 169–183, 1996.
6. Peter Borovanský. Controlling rewriting: study and implementation of a strategy formalism. *Electronic Notes in Theoretical Computer Science*, 15, 1998.
7. Peter Borovanský, Claude Kirchner, and Hélène Kirchner. A functional view of rewriting and strategies for a semantics of ELAN. In Masahiko Sato and Yoshihito Toyama, editors, *The Third Fuji International Symposium on Functional and Logic Programming*, pages 143–167, Kyoto, April 1998. World Scientific. Also report LORIA 98-R-165.
8. Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In *Proceedings of Workshop on Rewriting Techniques and Application - WRLA'1998*, volume 15. Electronic Notes in Theoretical Computer Science, September 1998. Report LORIA 98-R-316.
9. Mark van den Brand, Hayco. de Jong, Paul Klint, and Pieter Olivier. Efficient Annotated Terms. *Software, Practice & Experience*, 30:259–291, 2000.
10. Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. *Développement d'applications avec Objective CAML*. O'REILLY, 2000.
11. Horatiu Cirstea. *Calcul de réécriture : fondements et applications*. PhD thesis, Université Henri Poincaré - Nancy I, 2000. October 25.
12. Horatiu Cirstea, Germain Faure, and Claude Kirchner. A rho-calculus of explicit constraint application. In *Workshop on Rewriting Logic and Applications*, Barcelona (Spain), March 2004. Electronic Notes in Theoretical Computer Science.
13. Horatiu Cirstea and Claude Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.
14. Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Matching Power. In *Proceedings of Rewriting Techniques and Applications RTA'2001*, http://www.springer.de/comp/lncs/index.htmlLecture Notes in Computer Science, Utrecht (The Netherlands), May 2001. Springer-Verlag.
15. Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Rewriting calculus with(out) types. In Fabio Gadducci and Ugo Montanari, editors, *Proceedings of the fourth workshop on rewriting logic and applications*, Pisa (Italy), September 2002. Electronic Notes in Theoretical Computer Science.
16. Horatiu Cirstea, Claude Kirchner, Luigi Liquori, and Benjamin Wack. Rewrite strategies in the rewriting calculus. In *Proceedings of the Third International Workshop on Reduction Strategies in Rewriting and Programming* , Valencia, Spain, June 2003. Electronic Notes in Theoretical Computer Science.
17. Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In José Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

18. Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM (JACM)*, 43(2):362–397, 1996.
19. René David and Bruno Guillaume. A λ-calculus with explicit weakening and explicit substitution. *Mathematical Structures for Computer Science*, 11:169–206, 2001.
20. Nicolas de Bruijn. Lambda calculus with namefree formulas involving symbols that represent reference transforming mappings. *Indagationes Mathematicae*, 40:348–356, 1978.
21. Nicolas G. de Bruijn. A lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.
22. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher order unification via explicit substitutions. *Information and Computation*, 157(1/2):183–235, 2000.
23. Steven Eker. Associative-commutative matching via bipartite graph matching. *Computer Journal*, 38(5):381–399, 1995.
24. Germain Faure and Claude Kirchner. Exceptions in the rewriting calculus. In *Proceedings of Rewriting Techniques and Applications - RTA'2002*, volume 2378 of *LNCS*, pages 66–82, Copenhagen, July 2002. Springer-Verlag.
25. Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, New York (NY, USA), 1979.
26. Dimitri Hendriks and Vincent van Oostrom. The $\lambda\!\!\downarrow$ calculus. In *Proceedings of Conference on Automated Deduction - CADE'2003*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 136–150, 2003.
27. Paul Hudak and Joseph H. Fasel. A gentle introduction to Haskell. *ACM SIGPLAN Notices*, 27(5):T–1–T–53, May 1992.
28. Gérard Huet. *Resolution d'Equations dans les Languages d'Ordre 1,2,...,ω*. These de Doctorat D'Etat, Universite Paris VII, 1976.
29. Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
30. Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.
31. Delia Kesner and Stephane Lengrand. Extending the explicit substitution paradigm. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA '05)*, *LNCS 256*, 2005.
32. Claude Kirchner and Hélène Kirchner. Rewriting, solving, proving. A preliminary version of a book available at `www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.
33. Claude Kirchner and Hélène Kirchner. Rule-based programming and proving : the ELAN experience outcomes. In *Ninth Asian Computing Science Conference - ASIAN'04, Chiang Mai, Thailand*, Dec 2004.
34. Claude Kirchner, Pierre-Etienne Moreau, and Antoine Reilles. Formal validation of pattern matching code. In *PPDP '05: Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 187–197, New York, NY, USA, 2005. ACM Press.
35. Jan Willem Klop. *Combinatory Reduction Systems*. Ph.D. thesis, Mathematisch Centrum, Amsterdam, 1980.
36. Xavier Leroy. Compiling functional languages. *Summer school on Semantics of programming languages*, 2002.
37. Pierre Lescanne. From $\lambda\sigma$ to λv a journey through calculi of explicit substitutions. In *Conference Record of POPL '94*, pages 60–69. ACM, January 1994.
38. Chuck Liang and Gopalan Nadathur. Choices in representation and reduction strategies for lambda terms in intensional contexts. *Journal of Automated Reasoning*, 33(2):89–132, 2004.
39. Luigi Liquori and Bernard Serpette. irho: an imperative rewriting calculus. In *Proceedings of te International Conference on Principles and Practice of Declarative Programming (PPDP'04)*, pages 167–178, 2004.
40. Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, May 2003.
41. César Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types.* Thèse de doctorat, Université Paris 7, 1997. English version available as INRIA research report RR-3309.
42. Gopalan Nadathur. The suspension notation for lambda terms and its use in metalanguage implementations. *Ninth Workshop on Logic, Language, Information and Computation (WoLLIC'02)*, *Electronic Notes in Theoretical Computer Science*, 67, 2002.
43. Quang-Huy Nguyen, Claude Kirchner, and Hélène Kirchner. External rewriting for skeptical proof assistants. *Journal of Automated Reasoning*, 29(3-4):309–336, 2002.
44. Enno Ohlebusch. Church-rosser theorems for abstract reduction modulo an equivalence relation. In *Proceedings of Rewriting Techniques and Applications (RTA-98)*, volume 1379 of *LNCS*, pages 17–31. Springer, 1998.
45. Bruno Pagano. X.R.S: Explicit reduction systems - A first-order calculus for higher-order calculi. In *Proceedings of the International Conference on Automated Deduction (CADE '98)*, pages 72–87, 1998.
46. Kristoffer Høgsbro Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, University of Copenhagen, Denmark, February 1996.
47. Francois-Régis Sinot. Director strings revisited: A generic approach to the efficient representation of free variables in higher-order rewriting. *J. Log. Comput*, 15(2):201–218, 2005.
48. Mark-Oliver Stehr. CINNI - A generic calculus of explicit substitutions and its application to lambda-, varsigma- and pi- calculi. *Electronic Notes in Theoretical Computer Science*, 36, 2000.
49. Aaron Stump, Arun Deivanayagam, Spencer Kathol, Dylan Lingelbach, and Daniel Schobel. Rogue decision procedures. In *1st International Workshop on Pragmatics of Decision Procedures in Automated Reasoning - PDPAR 2003*, 2003.
50. Benjamin Wack. *Aspects logique du calcul de réécriture*. PhD thesis, Université Henri Poincaré - Nancy I, October 2005.
51. Hirofumi Yokouchi and Teruo Hikita. A rewriting system for categorical combinators with multiple arguments. *SIAM Journal on Computing*, 19(1):78–97, February 1990.