Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 176 (2007) 95–111
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Distributive ρ-calculus

Horatiu Cirstea[b], Clément Houtmann[a] and Benjamin Wack[b]

[a] *LORIA, ENS-Cachan, 61 avenue du Président Wilson, 94235 Cachan Cedex, France*
Clement.Houtmann@loria.fr

[b] *LORIA & NANCY I & NANCY II, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex France*
{Horatiu.Cirstea, Benjamin.Wack}@loria.fr

Abstract

The *rewriting calculus* has been introduced as a general formalism that uniformly integrates rewriting and λ-calculus. In this calculus all the basic ingredients of rewriting such as *rewrite rules*, *rule applications* and *results* are first-class objects. The rewriting calculus has been originally designed and used for expressing the semantics of rule based as well as object oriented paradigms. We have previously shown that convergent term rewriting systems and classic strategies can be encoded naturally in the calculus.
In this paper, we go a step further and we propose an extended version of the calculus that allows one to encode unrestricted term rewriting systems. This version of the calculus features a new evaluation rule describing the behavior of the result structures and a *call-by-value* evaluation strategy. We prove the confluence of the obtained calculus and the correctness and completeness of the proposed encoding.

Keywords: rewriting calculus, lambda calculus, term rewriting systems, fixpoints.

1 Introduction

The ability to discriminate patterns is one of the main basic mechanisms the human reasoning is based on. Indeed, the ability to recognize patterns, *i.e.* pattern matching, is present since the beginning of information processing modeling. Instances of it can be traced back to pattern recognition and it has been extensively studied when dealing with strings [11], trees [9] or feature objects [1].

Pattern matching has also been widely used in functional programming (*e.g.* ML, Haskell, Scheme), logic programming (*e.g.* Prolog), rewrite based programming (*e.g.* ASF+SDF [14], ELAN [2], Maude [13], Obj* [8]), script programming (*e.g.* sed, awk). It has been generally considered as a convenient mechanism for expressing complex requirements about the argument of a function, more than a real computation paradigm.

The *rewriting calculus* [5,7] by unifying λ-calculus and rewriting, makes all the basic ingredients of rewriting explicit objects, in particular the notions of *rule application* and *result*. Its basic idea is to abstract on patterns instead of simple

doi:10.1016/j.entcs.2007.06.010

variables as in the λ-calculus, and then to produce terms such as $f(x) \twoheadrightarrow x$, that could be represented in a λ-style as $\lambda f(x).x$.

The rewriting calculus has been originally designed and used for expressing the semantics of rule based as well as object oriented paradigms [6]. Indeed, in rewriting calculus the term rewriting system (TRS) consisting of the rules $a \to b$ and $b \to c$ can be represented by the structure $a \twoheadrightarrow b \wr b \twoheadrightarrow c$ and its application to the constant a is encoded by the term $(a \twoheadrightarrow b \wr b \twoheadrightarrow c)\ a$, *i.e.* the application of the structure to the argument. This latter term reduces in the rewriting calculus to b. If we consider the structure $a \twoheadrightarrow b \wr a \twoheadrightarrow c$ consisting of two rules with overlapping left-hand sides, the application $(a \twoheadrightarrow b \wr a \twoheadrightarrow c)\ a$ evaluates to the structure $b \wr c$ that can be seen as the non-deterministic choice between the two terms b and c.

General term rewriting systems and classical guiding strategies have been encoded in the original rewriting calculus [5] by adding an additional operator that intuitively selects one of the elements from a set of results. We have shown that an equivalent operator can be encoded in the current version of the calculus but the encoding is limited in this case to convergent term rewriting systems [7].

We show in this paper that the previously proposed encoding can be extended to the general case, *i.e.* to arbitrary term rewrite systems. For this, a new evaluation rule that enriches the semantics of the structure operator is added and an evaluation strategy is enforced by imposing a certain discipline on the application of the evaluation rules. This strategy is defined syntactically using an appropriate notion of value and is used in order to recover the confluence of the calculus that is lost in the general case.

Roadmap In Section 2, we give the syntax and the evaluation semantics of the proposed calculus and we prove its confluence. Then in Section 3, we discuss the expressive power of the calculus. More precisely we propose an encoding of (*non-convergent*) term rewriting systems in the calculus. Finally in Section 4, we conclude and give some perspectives of this work.

2 The distributive ρ-calculus: ρ_d-calculus

We present here the syntax and the semantics of the proposed calculus as well as its main properties.

2.1 Syntax

We consider in what follows the meta-symbols "$\_ \twoheadrightarrow \_$" (abstraction operator), and "$\_ \wr \_$" (structure operator), and the (hidden) application operator. We assume that the application operator associates to the left, while the other operators associate to the right. The priority of the application is higher than that of "$\_ \twoheadrightarrow \_$" which is, in turn, of higher priority than the "$\_ \wr \_$". The symbols $A, B, C, \ldots$ range over the set $\mathcal{T}$ of terms, the symbols $x, y, z, \ldots$ range over the set $\mathcal{X}$ of variables ($\mathcal{X} \subseteq \mathcal{T}$), the symbols $a, b, c, \ldots, f, g, h$ and string built from them range over a set $\mathcal{K}$ of term constants ($\mathcal{K} \subseteq \mathcal{T}$). Finally, the symbols P, Q range over the set $\mathcal{P}$ of patterns,

($\mathcal{X} \subseteq \mathcal{P} \subseteq \mathcal{T}$). All symbols can be indexed. The symbol stk is a special constant denoting matching failures and whose semantics will be given in the next section. To denote a tuple of terms $A_1 \dots A_n$, we will use the vector notation $\overline{A}$. This notation will be used in combination with the application operator : $A\overline{B}$ means $(((AB_1)\dots)B_n)$.

The syntax of the basic rewriting calculus is inductively defined as follows:

$$\begin{array}{lll} \mathcal{P} & ::= & \mathcal{X} \mid \mathcal{K} \mid \mathcal{K}\,\overline{\mathcal{P}} \mid \mathsf{stk} \qquad \text{Patterns} \\ \mathcal{T} & ::= & \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\,\mathcal{T} \mid \mathcal{T} \wr \mathcal{T} \mid \mathsf{stk} \qquad \text{Terms} \end{array}$$

We call *algebraic* the patterns used in this version of the calculus and we usually denote a term of the form $(\dots((f\ A_1)\ A_2)\dots)\ A_n$ with $f \in \mathcal{K}$ by $f(A_1, A_2, \dots, A_n)$. A *linear* pattern is a pattern where every variable occurs at most once.

The *values* represent intuitively the terms that we do not need to evaluate and are inductively defined by:

$$\mathcal{V} \ ::= \ \mathcal{X} \mid \mathcal{K} \mid \mathcal{K}\,\overline{\mathcal{V}} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \qquad \text{Values}$$

These values can be extended to the so-called *structure values* and *stuck values*, which will restrict the applications of the evaluation rules (γ), (ρ) and (δ):

$$\begin{array}{lll} \mathcal{V}_\gamma & ::= & \mathcal{V} \mid \mathcal{V}_\gamma \wr \mathcal{V}_\gamma \qquad \text{Structure Values} \\ \mathcal{V}_{\rho\delta} & ::= & \mathcal{V} \mid \mathsf{stk} \qquad \text{Stuck Values} \end{array}$$

One can notice that the only potential redexes (*i.e.* applications of variables, abstractions or structures) in values are inside abstractions. In what follows the symbol V ranges over the set $\mathcal{V}$ of values, the symbol V^γ ranges over the set $\mathcal{V}_\gamma$ of structure values, the symbol $V^{\rho\delta}$ ranges over the set $\mathcal{V}_{\rho\delta}$ of stuck values. All these symbols can be indexed.

Definition 2.1 (Free and bound variables) *Given a term A, the sets of its free variables denoted $\mathcal{FV}(A)$ and bound variables denoted $\mathcal{BV}(A)$ are defined as follows:*

$$\begin{array}{rlrl} \mathcal{FV}(x) & \triangleq \{x\} & \mathcal{BV}(x) & = \emptyset \\ \mathcal{FV}(f) & \triangleq \emptyset & \mathcal{BV}(f) & = \emptyset \\ \mathcal{FV}(P \twoheadrightarrow A) & \triangleq \mathcal{FV}(A) \setminus \mathcal{FV}(P) & \mathcal{BV}(P \twoheadrightarrow A) & = \mathcal{BV}(A) \cup \mathcal{FV}(P) \\ \\ \mathcal{FV}(A\ B) & \triangleq \mathcal{FV}(A) \cup \mathcal{FV}(B) & \mathcal{BV}(A\ B) & = \mathcal{BV}(A) \cup \mathcal{BV}(B) \\ \mathcal{FV}(A \wr B) & \triangleq \mathcal{FV}(A) \cup \mathcal{FV}(B) & \mathcal{BV}(A \wr B) & = \mathcal{BV}(A) \cup \mathcal{BV}(B) \\ \mathcal{FV}(\mathsf{stk}) & \triangleq \emptyset & \mathcal{BV}(\mathsf{stk}) & = \emptyset \end{array}$$

As usual, we work modulo "α-*conversion*" and adopt Barendregt's "*hygiene-convention*" , *i.e.* free and bound variables have different names.

Definition 2.2 (Substitutions)
A substitution θ is a mapping from the set of variables to the set of terms. A finite substitution θ has the form $\{A_1/x_1 \dots A_m/x_m\}$, and its domain $\{x_1, \dots, x_m\}$ is denoted by $\mathcal{D}om(\theta)$. The application of a substitution θ to a term A such that $\mathcal{D}om(\theta) \cap \mathcal{BV}(A) = \emptyset$, denoted by $A\theta$, is defined as follows:

$$x_i\theta \triangleq \begin{cases} A_i \text{ if } x_i \in \mathcal{D}om(\theta) \\ x_i \text{ otherwise} \end{cases} \qquad (P \twoheadrightarrow A)\theta \triangleq P \twoheadrightarrow A\theta$$

$$f\theta \triangleq f \qquad (A\ B)\theta \triangleq A\theta\ B\theta$$

$$\mathsf{stk}\theta \triangleq \mathsf{stk} \qquad (A \wr B)\theta \triangleq A\theta \wr B\theta$$

We should point out that since we consider classes of terms modulo the α-conversion, any term A has a proper representative A' such that $\mathcal{BV}(A') \cap \mathcal{D}om(\theta) = \emptyset$, which avoids potential variable captures.

2.2 Operational semantics

The evaluation mechanism of the rewriting calculus relies on the fundamental operation of *matching* that allows us to bind variables to their current values. In the general rewriting 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 [6].

For the purpose of this paper we restrict to syntactic matching, in which case the matching substitution, when it exists, is unique and can be computed by a simple recursive algorithm given for example by G. Huet [10].

The operational semantics of the ρ_d-calculus is defined by the following rules:

$$(P \twoheadrightarrow A)\ V^{\rho\delta} \to_p A\theta \qquad \text{if } P\theta \equiv V^{\rho\delta}$$

$$(A \wr B)\ V^{\rho\delta} \to_\delta A\ V^{\rho\delta} \wr B\ V^{\rho\delta}$$

$$A\ (V_1^\gamma \wr V_2^\gamma) \to_\gamma A\ V_1^\gamma \wr A\ V_2^\gamma$$

The rule (ρ) can be applied if (and only if) such a substitution θ exists and in this case it is applied to the term A. If such a substitution does not exist then this rule can not be fired and the term is left as it is, representing a failure. Nevertheless, further reductions or instantiations are likely to modify $V^{\rho\delta}$ so that the appropriate

substitution can be found and the rule can be fired. The rule (δ) right-distributes the application over the structures. This gives the possibility, for example, to apply in parallel two distinct pattern abstractions to a given term. The rule (γ) is the counterpart of the rule (δ) and left-distributes the application of a term over a structure. The implicit conditions imposing that the arguments of an application are values are essentially related to the confluence of the calculus and are discussed in Section 2.3.

Definition 2.3 (One-step relation)

The one-step relation induced by a set of rewrite rules $\mathcal{R}$ is noted $\mapsto_{\mathcal{R}}$ and is the compatible closure of the relation induced by the set of rules $\mathcal{R}$:

- *if $t \to_{\mathcal{R}} u$ then $t \mapsto_{\mathcal{R}} u$;*
- *if $t \mapsto_{\mathcal{R}} u$ then $f(t_1, \ldots, t, \ldots, t_n) \mapsto_{\mathcal{R}} f(t_1, \ldots, u, \ldots, t_n)$.*

The multi-step relation, denoted $\mapsto\!\!\!\!\to_{\mathcal{R}}$, is the reflexive and transitive closure of $\mapsto_{\mathcal{R}}$. Similarly, the multi-step relation induced by the rules of the ρ_d-calculus is denoted $\mapsto\!\!\!\!\to_{\rho\delta\gamma}$, with the compatible closure defined as follows.

Definition 2.4 (Compatible closure of $\to_{\rho\delta\gamma}$)

In the distributive ρ-calculus, a context *is a special term defined by the following grammar:*

$$C[\,] \quad ::= \quad [\,] \mid \mathcal{P} \twoheadrightarrow C[\,] \mid \mathcal{T}\, C[\,] \mid C[\,]\, \mathcal{T} \mid C[\,] \wr \mathcal{T} \mid \mathcal{T} \wr C[\,]$$

The compatible closure of $\to_{\rho\delta\gamma}$ is the (finest) relation $\mapsto_{\rho\delta\gamma}$ such that if $t \to_{\rho\delta\gamma} u$, then for any context $C[\,]$, we have $C[t] \mapsto_{\rho\delta\gamma} C[u]$.

Example 2.5 (Simple example + failures)

If we consider the terms $(f(x) \twoheadrightarrow (3 \twoheadrightarrow 3)\ x)\ f(3)$ and $(f(x) \twoheadrightarrow (3 \twoheadrightarrow 3)\ x)\ f(4)$ then the following reductions are obtained:

$$(f(x) \twoheadrightarrow (3 \twoheadrightarrow 3)\ x)\ f(3) \quad \mapsto_{\rho} \quad (3 \twoheadrightarrow 3)\ 3 \quad \mapsto_{\rho} \quad 3$$

$$(f(x) \twoheadrightarrow (3 \twoheadrightarrow 3)\ x)\ f(4) \quad \mapsto_{\rho} \quad (3 \twoheadrightarrow 3)\ 4$$

The term $(a \twoheadrightarrow b \wr a \twoheadrightarrow c)\ a$ reduces to $b \wr c$:

$$(a \twoheadrightarrow b \wr a \twoheadrightarrow c)\ a \quad \mapsto_{\delta} \quad (a \twoheadrightarrow b)\ a \wr (a \twoheadrightarrow c)\ a \quad \mapsto\!\!\!\!\to_{\rho} \quad b \wr c$$

The term $(a \twoheadrightarrow b \wr b \twoheadrightarrow c)\ a$ reduces similarly to $b \wr (b \twoheadrightarrow c)\ a$.

Notice that the term $(a \twoheadrightarrow b \wr b \twoheadrightarrow c)\ a$ does not reduce to b as one might expect. Instead, the fact that the rule $b \twoheadrightarrow c$ fails to apply to a (in classical rewriting) is also recorded in the final result as a (failure) term in normal form. This approach is very interesting when we want to handle explicitly the failures by allowing rules that can handle such particular terms (*e.g.* for an exception handling mechanism). However, if the user is not interested in the explicit manipulation of matching failures and just wants to ignore such a behavior, we need to handle uniformly matching failures and eliminate them when not significant for the computation.

For this, we first want to represent all the definitive failures by the constant `stk` whose exact semantics should be the following: if for any reduction of the argument,

there exist no matching substitution, then the ρ-redex is reduced to stk:

$$\frac{\forall \theta_1, \theta_2,\ \forall B',\ B\theta_1 \mapsto\!\!\!\!\!\rightarrow_{\rho\delta\gamma} B' \ \Rightarrow\ P\theta_2 \not\equiv B'}{(P \twoheadrightarrow A)\, B \rightarrow_{\mathsf{stk}} \mathsf{stk}}$$

One can easily notice that B can contain a ρ-term with an arbitrary (possibly infinite) number of possible reductions which should be all explored in order to decide if the appropriate substitution exists. The condition of this rule is thus undecidable and consequently the operational semantics of the calculus cannot be defined using such a rule. Nevertheless, in practice and particularly when dealing with term rewriting systems we do not need to be so general and a sufficient condition can be used.

Definition 2.6 (Definitive failures)
The relation $\not\sqsubseteq$ on $\mathcal{P} \times \mathcal{T}$ is inductively defined by:

$$\begin{array}{lll}
\mathsf{stk} & \not\sqsubseteq g\,\overline{B} & \text{if } g \not\equiv \mathsf{stk} \\
\mathsf{stk} & \not\sqsubseteq Q \twoheadrightarrow B & \\
f\,P_1 \ldots P_m & \not\sqsubseteq g\,B_1 \ldots B_n & \text{if } f \not\equiv g \text{ or } n \neq m \text{ or } \exists i,\ P_i \not\sqsubseteq B_i \\
f\,\overline{P} & \not\sqsubseteq \mathsf{stk} & \\
f\,\overline{P} & \not\sqsubseteq Q \twoheadrightarrow B &
\end{array}$$

Starting from this relation, the operational semantics of the ρ_d^{stk}-calculus are defined by the rules (ρ), (δ), (γ) introduced above and by the following rules:

$$\begin{array}{lll}
(P \twoheadrightarrow A)\, B \rightarrow_{\mathsf{stk}} \mathsf{stk} & \quad \text{if } P \not\sqsubseteq B \\
\mathsf{stk} \wr A \rightarrow_{\mathsf{stk}} A & \\
A \wr \mathsf{stk} \rightarrow_{\mathsf{stk}} A & \\
\mathsf{stk}\, A \rightarrow_{\mathsf{stk}} \mathsf{stk} &
\end{array}$$

As mentioned previously, these rules are used to determine, propagate or eliminate the definitive failures. If the matching between the left-hand side of a rule and the argument the rule is applied on is definitive then the failure is made explicit by transforming the application into a stk; this is done by the first rule. Structures can be seen as collections of results and thus we want to eliminate all the (matching) failures from these collections; this is done by the next two rules. On the other hand, a stk term can be seen as an empty set of results; the last rule corresponds then to the (δ) rule dealing with empty structures and thus, to a propagation of the failure. We will see in Section 3 why the stk-rule corresponding to the (γ) rule is not suitable.

The $\rightarrow_{\mathsf{stk}}$ induced relations are denoted $\mapsto_{\mathsf{stk}}$, $\mapsto\!\!\!\!\!\rightarrow_{\mathsf{stk}}$. The relation $\mapsto_{\rho\delta\gamma} \cup \mapsto_{\mathsf{stk}}$ is

denoted $\mapsto^{\mathsf{stk}}_{\rho\delta\gamma}$ and its transitive and reflexive closure is denoted $\mapsto\!\!\!\!\rightarrow^{\mathsf{stk}}_{\rho\delta\gamma}$.

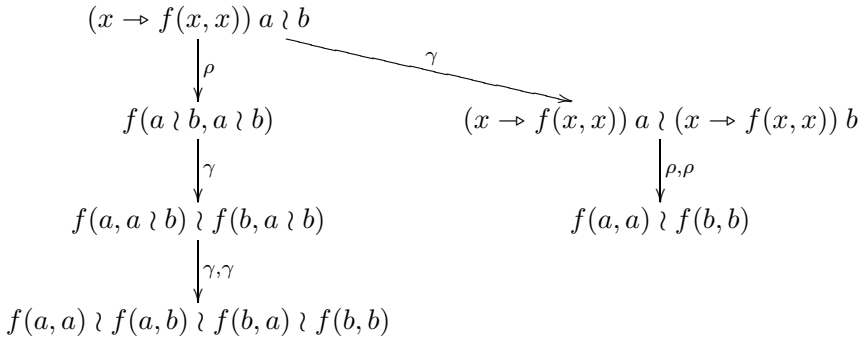
Example 2.7 (failures)
The term $(a \twoheadrightarrow b \wr b \twoheadrightarrow c)\, a$ *reduces now to* b*:*

$$(a \twoheadrightarrow b \wr b \twoheadrightarrow c)\, a \;\mapsto_{\delta}\; (a \twoheadrightarrow b)\, a \wr (b \twoheadrightarrow c)\, a \;\mapsto_{p}\; b \wr (b \twoheadrightarrow c)\, a \;\mapsto_{\mathsf{stk}}\; b \wr \mathsf{stk} \;\mapsto_{\mathsf{stk}}\; b$$

2.3 Properties

As we have mentioned in the previous section the ρ_d-calculus would not be confluent if we did not restrict the application of an abstraction and of a structure to be effective only when the argument is a value. When this restriction is not imposed on the (ρ) rule, potentially non-joinable critical pairs between the rules (ρ) and (γ) are obtained. Intuitively, restricting the argument of the application in the rule (ρ) to a value guarantees that it has been reduced enough to check if there exists a unique match between the pattern and the argument. Alternatively, we can accept any term as argument and use a more complex matching algorithm to find the appropriate substitution.

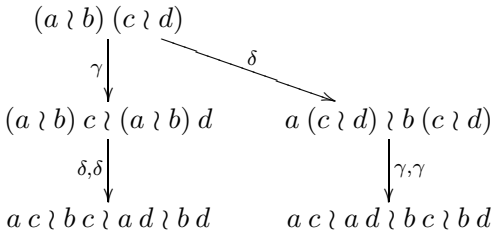
Example 2.8 (ρ without values)
When the conditions on values in the rule (ρ) *are omitted, non-confluent reductions can be obtained:*



Similarly, when the argument of the application is not restricted to a value in the (δ) rule, a critical pair between the rules (δ) and (γ) is obtained. The confluence can be retrieved either by enforcing this condition or by using an associative-commutative underlying theory for the structure operator.

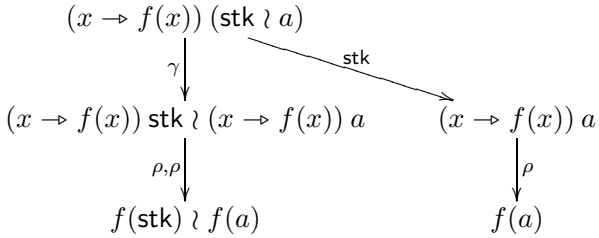
Example 2.9 (δ without values)
When no conditions are imposed in the rule (δ) *non-confluent reductions can be*

obtained:



For the (γ) rule, the condition imposes that the terms in the structure do not reduce to a failure. If one of them can lead to a failure then it should be first reduced to stk and then eliminated from the structure using the stk rules.

Example 2.10 (γ without values)
If the terms of a structure applied to an argument are not restricted to values then the application of the rule (γ) can lead to non-confluent reductions:



It is quite clear that using a set of values leads to a *call-by-value* reduction strategy. The two calculi presented above are confluent in this case.

Theorem 2.11 (Confluence of left-linear ρ_d-calculus)
If all patterns are linear, the relation $\mapsto_{\rho\delta\gamma}$ is confluent.

Proof. The proof is detailed in [4]. It uses the parallel reduction technique introduced for the λ-calculus in [12]. □

Theorem 2.12 (Confluence of left-linear ρ_d^{stk}-calculus)
If all patterns are linear, the relation $\mapsto^{\mathsf{stk}}_{\rho\delta\gamma}$ is confluent.

Proof. The proof is detailed in [4]. It is based on the proof introduced in [7,16].□

The unrestricted ρ_d-calculus is non-confluent since the Klop counter example holds in this case (see [4]).

2.4 ρ_d-calculus modulo some congruence

Defining a similar calculus *modulo* some congruence is certainly interesting but out of the scope of this paper. Such an extension would induce an encoding of general term rewriting systems just as our present calculus induces the encoding of syntactic term rewriting systems presented in the next section.

The main difficulty in defining such a calculus comes from the fact that matching modulo the given congruence is generally non-unitary (at least for classical theories like associativity and commutativity). Indeed there might exist (infinitely) many solutions and there exists no natural ordering for these solutions (*i.e.* substitutions).

For example let us consider the term $(f(x,y) \twoheadrightarrow x)\ f(a,b)$ when working modulo the commutativity of the symbol f. There exist two solutions of the matching problem: $\{a/x, b/y\}$ and $\{b/x, a/y\}$. Depending on the substitution we use, we obtain two possible non-confluent reductions:

$$\begin{array}{ccc} (f(x,y) \twoheadrightarrow x)\ f(a,b) & \overset{C}{=\!=} & (f(y,x) \twoheadrightarrow y)\ f(a,b) \\ \downarrow \rho & & \downarrow \rho \\ a \wr b & & b \wr a \end{array}$$

In order to recover confluence, the only solution may finally consist in declaring the structure operator as commutative, associative and idempotent.

Moreover, a particular reduction strategy should be enforced when working modulo some congruence. This strategy should prevent matching against uninstantiated terms, which would lead to non-confluent reductions as shown in the following example where the symbol "::" is considered associative:

$$\begin{array}{ccc} (z \twoheadrightarrow ((x :: y \twoheadrightarrow x)\ (a :: z)))\ (b :: c) & & \\ \downarrow \rho \qquad\qquad \searrow \rho & & \\ (x :: y \twoheadrightarrow x)\ (a :: (b :: c)) & & ((z \twoheadrightarrow a)\ (b :: c)) \\ \downarrow \rho & & \downarrow \rho \\ a \wr (a :: b) & & a \end{array}$$

Finally, the notion of *definitive failures* should be adapted to the reduction strategy of the new calculus in order to guarantee the coherence with the considered matching.

3 Encoding Term Rewriting Systems

We have already shown [6,16] that (the reduction of) *convergent* term rewriting systems (TRS) can be encoded in the classical rewriting calculus. The restriction to convergent TRS is due to the "uncomplete" treatment of the structure operator in the classical rewriting calculus where the application operator is left-distributive over the structure operator but not right-distributive. As we have already seen this choice was motivated by the meta-properties the calculus should have. More precisely, adding right-distributivity would lead to a non-confluent calculus. Nevertheless, this property can be retrieved either by enforcing a certain discipline on the evaluation (strategy) [5] or by restricting the term formation as done in this paper.

In ρ_d^{stk}-calculus the (γ) rule defines the right-distributivity of the application over the structure and in this section we show how this feature can be used to encode (non-confluent) TRS in the ρ_d^{stk}-calculus.

More precisely, given a TRS $\mathcal{R}$ we build the terms $\Omega^1_{\mathcal{R}}$ and $\Omega_{\mathcal{R}}$ such that

- $\Omega^1_{\mathcal{R}}\ m$ represents (*i.e.* reduces to) the one-step reduced of m *w.r.t.* $\mathcal{R}$,
- $\Omega_{\mathcal{R}}\ m$ represents the normal form of m *w.r.t.* $\mathcal{R}$ (if it exists).

3.1 Rule selection

As we wish to compute the normal forms, we obviously wish to decide when the reduction is effective, *i.e.* when some rule of $\mathcal{R}$ can be applied, and then to discriminate cases:

- if some rule of $\mathcal{R}$ can be applied to m, then we reduce m,
- if not, m is a normal form, and m is left as it is.

This ability to discriminate cases, *i.e.* to select between two (or more) terms which one can be applied successfully to a given argument, is encoded in the first term usually defined [7] in the rewriting calculus by:

$$\mathsf{first} \triangleq u \twoheadrightarrow v \twoheadrightarrow x \twoheadrightarrow (\mathsf{stk} \twoheadrightarrow v\ x \wr y \twoheadrightarrow y)\ (u\ x)$$

One can easily check that first has the intended behavior:

- $(\mathsf{first}\ A_1\ A_2)\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} V_1^{\rho\delta}$ if $A_1\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} V_1^{\rho\delta}$;
- $(\mathsf{first}\ A_1\ A_2)\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} V_2^{\rho\delta}$ if $A_1\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} \mathsf{stk}$ and $A_2\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} V_2^{\rho\delta}$.

Intuitively, if we replace the term A_1 by the ρ-term R encoding a TRS $\mathcal{R}$ and the term A_2 by the identity then we obtain the desired discrimination facility: the case $R\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} V_1^{\rho\delta}$ corresponds to a reduction of t *w.r.t.* $\mathcal{R}$ while $R\ t \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} \mathsf{stk}$ corresponds to the case where no rule can be applied to t and thus the term is left as it is (in fact the identity is applied to this term).

As the normal form of some terms *w.r.t.* a non-confluent TRS is not unique, we will obviously have to deal with *sets* of results. We choose here to encode sets of results as structures. The empty set is represented by stk and the union of two sets is represented using the structure operator. In the rewriting calculus the representation of some set is not unique as the structure operator is not considered as commutative, associative or idempotent.

Since we wish to discriminate cases such as *no rule of* $\mathcal{R}$ *matches* m, reformulated as *the set of one-step reduced of* m *is empty*, we need to pattern match on stk. The statement *if the set* M *is empty, then* T_1 *else* T_2 can be encoded by

$$\mathsf{first}\ (\mathsf{stk} \twoheadrightarrow T_1)\ (x \twoheadrightarrow T_2)\ M$$

Since we need the ability to pattern match on stk we have the rule

$$\mathsf{stk}\ A \to_{\mathsf{stk}} \mathsf{stk}$$

that complements the (δ) rule but not the symmetric one

$$A\ \mathsf{stk} \to_{\mathsf{stk}} \mathsf{stk}$$

that would complement the (γ) rule and that would correspond to a *strict* propagation of the failure. [1]

3.2 Context propagation

When rewriting *w.r.t.* a rewriting system, the application of the rules can be done on any *subterm* of the rewritten term. In the rewriting calculus, a rule is always applied on the head of the term and thus the encoding of a TRS has to propagate explicitly the application deeper in the term. For example, the application of the rewrite rule $a \to b$ to the term $f(a)$ is naively encoded by the term $(f(x) \twoheadrightarrow f((a \twoheadrightarrow b)\ x))\ f(a)$ that eventually reduces, as expected, to $f(b)$.

If the application of a rewrite rule fails on all the subterms of a given term then the ρ-term encoding the application should be reduced to stk. On the other hand, if we apply the same naive methodology as above for propagating the rule application into contexts then the application of the rewrite rule $a \to b$ to the term $f(b)$ is encoded by the term $(f(x) \twoheadrightarrow f((a \twoheadrightarrow b)\ x))\ f(b)$ that reduces to $f(\mathsf{stk})$ and not to stk.

More generally, the propagation of stk should be performed *w.r.t.* to any context. Therefore, for each symbol f of arity $n \geqslant 1$ from a signature Σ we define a term Γ_k^f:

$$\Gamma_k^f \triangleq \nu \twoheadrightarrow \overline{x}^n \twoheadrightarrow (nop(z) \twoheadrightarrow z)\ (\mathsf{first} \begin{pmatrix} \mathsf{stk} \twoheadrightarrow nop(\mathsf{stk}) \\ y \twoheadrightarrow nop(f(x_1, \ldots, x_{k-1}, y, \ldots, x_n)) \end{pmatrix} (\nu\ x_k))$$

where $nop \notin \Sigma$ and for any $n \geqslant 1$, $\overline{x}^n \twoheadrightarrow M \triangleq x_1 \twoheadrightarrow x_2 \twoheadrightarrow \ldots \twoheadrightarrow x_n \twoheadrightarrow M$.

Each Γ_k^f allows us to express the application of a given term to the subterm M_k of some term $f(M_1, \ldots, M_n)$. The following lemma states the behavior of Γ_k^f:

Lemma 3.1 *Let $f \in \Sigma$ be a symbol of arity n. Let $M_1, \ldots, M_n$ be some algebraic terms and T an arbitrary term. Let $V_1^\gamma, \ldots V_p^\gamma$ be some values in $\mathcal{V}_\gamma$. Then*

$$\Gamma_k^f\ T\ M_1 \ldots M_n \mapsto\!\!\!\!\!\rightarrow_{\rho\hat{\gamma}}^{\mathsf{stk}} \begin{pmatrix} f(M_1, \ldots, M_{k-1}, V_1^\gamma, \ldots M_n) \\ \wr \ldots \\ \wr\ f(M_1, \ldots, M_{k-1}, V_p^\gamma, \ldots, M_n) \end{pmatrix}$$

if $T\ M_k \mapsto\!\!\!\!\!\rightarrow_{\rho\hat{\gamma}}^{\mathsf{stk}} V_1^\gamma \wr \ldots \wr V_p^\gamma$ and
$\Gamma_k^f\ T\ M_1 \ldots M_n \mapsto\!\!\!\!\!\rightarrow_{\rho\hat{\gamma}}^{\mathsf{stk}} \mathsf{stk}$ if $T\ M_k \mapsto\!\!\!\!\!\rightarrow_{\rho\hat{\gamma}}^{\mathsf{stk}} \mathsf{stk}$.

Proof. The proof of this lemma just consists in checking that the reductions hold. It is presented in [4]. □

Let us remark that for any patterns P_1 and P_2 the term $\mathsf{first}\ (P_1 \twoheadrightarrow \mathsf{stk})\ (P_2 \twoheadrightarrow M)\ N$ will always reduce to the same term as $(P_2 \twoheadrightarrow M)\ N$. Indeed the first operator does not check if N matches P_1 but if $(P_1 \twoheadrightarrow \mathsf{stk})\ N$ reduces to stk which

[1] This strict behavior can be obviously encoded using the rule $\mathsf{stk} \twoheadrightarrow \mathsf{stk}$.

is always the case. Consequently, the term

$$\nu \twoheadrightarrow \overline{x}^n \twoheadrightarrow (\mathsf{first} \begin{pmatrix} \mathsf{stk} \twoheadrightarrow \mathsf{stk} \\ y \twoheadrightarrow f(x_1, \ldots, x_{k-1}, y, \ldots, x_n) \end{pmatrix} (\nu\ x_k))$$

does not have the same behavior as Γ^f_k. The use of the constant *nop* in this latter term allows us to claim that a reduction to stk is equivalent to a pattern matching failure.

We can now define the term Γ^f

$$\Gamma^f \triangleq \Gamma^f_1 \wr \ldots \wr \Gamma^f_n$$

that represents intuitively the application of some term to each subterm M_k of a term $M = f(M_1, \ldots M_n)$. The structure grouping together the different results obtained when a term T is applied to M is obtained by reducing the ρ-term $\Gamma^f\ T\ M$:

$$\Gamma^f\ T\ M \mapsto\!\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\mathcal{S}\gamma} \Gamma^f_1\ T\ M \wr \ldots \wr \Gamma^f_n\ T\ M$$

3.3 One-step reduction

Let us consider now a term rewriting system $\mathcal{R} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$. We denote by $\mapsto_{\mathcal{R}}$ the compatible closure of $\mathcal{R}$, $\mapsto\!\!\!\!\!\twoheadrightarrow_{\mathcal{R}}$ its transitive and reflexive closure. The multiset of all one-step reducts of a term M is denoted $\{T \mid M \mapsto_{\mathcal{R}} T\}$ where the arity of some term T is the number of one-step reductions from M to T. Finally we write $M \mapsto\!\!\!\!\!\twoheadrightarrow_{\mathcal{R}!} T$ if and only if $M \mapsto\!\!\!\!\!\twoheadrightarrow_{\mathcal{R}} T$ and there exists no term N such that $T \mapsto_{\mathcal{R}} N$. The multiset of all normal forms of a term M *w.r.t.* $\mathcal{R}$ is denoted $\{T \mid M \mapsto\!\!\!\!\!\twoheadrightarrow_{\mathcal{R}!} T\}$ where the arity of some term T is the number of multi-step reductions from M to T.

The term that encodes the one-step reduction *w.r.t.* a term rewrite system $\mathcal{R}$ is denoted by $\Omega^1_{\mathcal{R}}$ and defined by

$$\Omega^1_{\mathcal{R}} \triangleq \omega^1_{\mathcal{R}}\ \omega^1_{\mathcal{R}}$$

where $\omega^1_{\mathcal{R}} \triangleq$

$$\pi \twoheadrightarrow \begin{pmatrix} \ldots \wr\ l_i \twoheadrightarrow r_i \wr \ldots \wr & \text{for all } l_i \to r_i \in \mathcal{R} \\ \ldots \wr\ f(x_1, \ldots, x_n) \twoheadrightarrow \Gamma^f\ (\pi\ \pi)\ x_1 \ldots x_n \wr \ldots & \text{for all } f \text{ of arity } n \geqslant 1 \end{pmatrix}$$

The definition of $\omega^1_{\mathcal{R}}$ can be cut in two parts: the first one encodes the rewriting at the head position *w.r.t.* $\mathcal{R}$ since we only transcript each rule of $\mathcal{R}$ by the corresponding rule in the ρ^{stk}_d-calculus; the second one uses the terms Γ^f_k to express that we also rewrite inside contexts. The term $\Omega^1_{\mathcal{R}}$ completes a fixpoint by means of the expression $(\pi\ \pi)$ in order to iterate the use of the Γ^k_f and to get down in the term as much as needed.

Theorem 3.2 *Let M be an algebraic term.*

- *If* $\{T \mid M \mapsto_{\mathcal{R}} T\} = \emptyset$ *then*
$$\Omega^1_{\mathcal{R}}\, M \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} \mathsf{stk}\ .$$
- *If* $\{T \mid M \mapsto_{\mathcal{R}} T\} \neq \emptyset$ *then*

$$\Omega^1_{\mathcal{R}}\, M \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} T_1 \wr \ldots \wr T_p \text{ with } \{T \mid M \mapsto_{\mathcal{R}} T\} = \{T_1, \ldots T_p\}\ .$$

Moreover as the left-linear ρ^{stk}_d-calculus is confluent, if $\mathcal{R}$ is left-linear and since $T_1 \wr \ldots \wr T_p$ and stk *are in normal form, then these are the unique normal forms of $\Omega^1_{\mathcal{R}}\, M$.*

Proof. The proof of this Theorem is done by induction on the term M. It is presented in [4]. □

3.4 Normal form reduction

We now define the term that encodes the normal form reduction *w.r.t.* a term rewrite system $\mathcal{R}$.

More precisely, we want to define a term $\Omega_{\mathcal{R}}$ such that its application to some term M, $\Omega_{\mathcal{R}} M$ reduces to M if $\Omega^1_{\mathcal{R}} M$ reduces to stk (M is a normal form) and continues applying the term $\Omega_{\mathcal{R}}$ to the result of $\Omega^1_{\mathcal{R}} M$ if it is different from stk. We define thus the term

$$\Omega_{\mathcal{R}} \triangleq \omega_{\mathcal{R}}\, \omega_{\mathcal{R}}$$

where

$$\omega_{\mathcal{R}} \triangleq s \twoheadrightarrow x \twoheadrightarrow \mathsf{first}\ (\mathsf{stk} \twoheadrightarrow x)\ (z \twoheadrightarrow (s\ s)\ z)\ (\Omega^1_{\mathcal{R}}\ x)$$

Let us introduce now the relation $\mathrel{-\!\!\in}$ that represents intuitively the observability of some "result" in a structured set of terms.

Definition 3.3 The relation $\mathrel{-\!\!\in}$ is defined inductively by:

- for any term M, $M \mathrel{-\!\!\in} M$;
- for any terms M, N_1 and N_2, $M \mathrel{-\!\!\in} N_1 \Rightarrow M \mathrel{-\!\!\in} N_1 \wr N_2$ and $M \mathrel{-\!\!\in} N_2 \wr N_1$.

Using the above relation we can state the correctness and completeness of the encoding:

Theorem 3.4 *Given two algebraic terms M and M',*

$$M \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{R}!} M' \quad \Longleftrightarrow \quad \exists T,\ \Omega_{\mathcal{R}} M \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} T \text{ and } M' \mathrel{-\!\!\in} T.$$

Moreover if $\mathcal{R}$ terminates on M then

$$\Omega_{\mathcal{R}}\, M \mapsto\!\!\!\!\twoheadrightarrow^{\mathsf{stk}}_{\rho\delta\gamma} T_1 \wr \ldots \wr T_p \text{ with } \{T \mid M \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{R}!} T\} = \{T_1, \ldots T_p\}\ .$$

Moreover as the left-linear ρ^{stk}_d-calculus is confluent, if $\mathcal{R}$ is left-linear and since $T_1 \wr \ldots \wr T_p$ is a normal form, it is the unique normal form of $\Omega_{\mathcal{R}}\, M$.

Proof. The proof of this theorem is done first by induction on the term M, and then by induction on the largest length of a reduction *w.r.t.* $\mathcal{R}$ of M to a normal form. It uses the Theorem 3.2 and it is presented in [4]. □

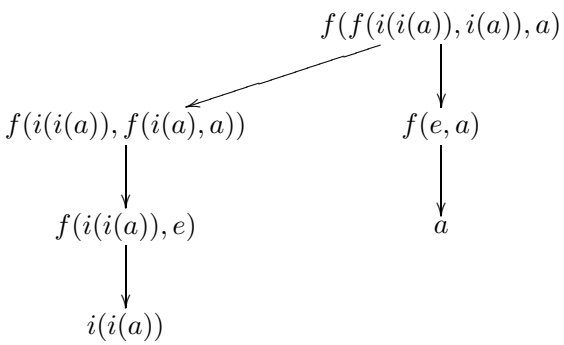
This theorem claims that our encoding of some term rewriting system encodes its reductions in the ρ_d^{stk}-calculus. Indeed the ρ_d^{stk}-calculus computes the finite multiset of normal forms of any term on which the term rewriting system terminates. Moreover if the system is divergent on some term, all reductions are still encoded since the ρ_d^{stk}-calculus computes a non-terminating reduction, generating normal forms as in a breadth first search of the reduction tree. All normal forms are computed at some iteration although the computation never stops, and may even never stop generating new normal forms.

Example 3.5 (Oriented groups) *Let us consider the group theory axioms oriented as follows:*

$$\Sigma = \{e(0),\ i(1),\ f(2)\}$$

$$\begin{cases} f(x,e) \to x \\ f(e,x) \to x \end{cases} \qquad \begin{cases} f(x,i(x)) \to e \\ f(i(x),x) \to e \end{cases} \qquad f(f(x,y),z) \to f(x,f(y,z))$$

This TRS is non-confluent since



The terms $\omega_{\mathcal{R}}^1$, $\Omega_{\mathcal{R}}^1$, $\omega_{\mathcal{R}}$ *and* $\Omega_{\mathcal{R}}$ *are then defined by:*

$$\omega^1_{\mathcal{R}} \triangleq$$

$$\pi \twoheadrightarrow \left(\begin{array}{l} f(x,e) \twoheadrightarrow x \\ f(e,x) \twoheadrightarrow x \\ f(x,i(x)) \twoheadrightarrow e \\ f(i(x),x) \twoheadrightarrow e \\ f(f(x,y),z) \twoheadrightarrow f(x,f(y,z)) \\ i(x) \twoheadrightarrow (nop(z) \twoheadrightarrow z)\ (\mathsf{first} \left(\begin{array}{l} \mathsf{stk} \twoheadrightarrow nop(\mathsf{stk}) \\ y \twoheadrightarrow nop(i(y)) \end{array} \right)\ ((\pi\ \pi)\ x)) \\ f(x_1,x_2) \twoheadrightarrow (nop(z) \twoheadrightarrow z)\ (\mathsf{first} \left(\begin{array}{l} \mathsf{stk} \twoheadrightarrow nop(\mathsf{stk}) \\ y \twoheadrightarrow nop(f(y,x_2)) \end{array} \right)\ ((\pi\ \pi)\ x_1)) \\ f(x_1,x_2) \twoheadrightarrow (nop(z) \twoheadrightarrow z)\ (\mathsf{first} \left(\begin{array}{l} \mathsf{stk} \twoheadrightarrow nop(\mathsf{stk}) \\ y \twoheadrightarrow nop(f(x_1,y)) \end{array} \right)\ ((\pi\ \pi)\ x_2)) \end{array} \right),$$

$$\Omega^1_{\mathcal{R}} \triangleq \omega^1_{\mathcal{R}}\ \omega^1_{\mathcal{R}},$$

$$\omega_{\mathcal{R}} \triangleq s \twoheadrightarrow x \twoheadrightarrow \mathsf{first}\ (\mathsf{stk} \twoheadrightarrow x)\ (z \twoheadrightarrow (s\ s)\ z)\ (\Omega^1_{\mathcal{R}}\ x)$$

and

$$\Omega_{\mathcal{R}} \triangleq \omega_{\mathcal{R}}\ \omega_{\mathcal{R}}.$$

Then we have the following reductions in the ρ^{stk}_d-calculus:

one-step reductions

$$\begin{array}{lll} \Omega^1_{\mathcal{R}}\ f(f(i(i(a)),i(a)),a) & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & f(i(i(a),f(i(a),a)) \wr f(e,a) \\ \Omega^1_{\mathcal{R}}\ f(i(i(a)),f(i(a),a)) & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & f(i(i(a)),e) \\ \Omega^1_{\mathcal{R}}\ f(i(i(a)),e) & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & i(i(a)) \\ \Omega^1_{\mathcal{R}}\ i(i(a)) & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & \mathsf{stk} \\ \Omega^1_{\mathcal{R}}\ f(e,a) & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & a \\ \Omega^1_{\mathcal{R}}\ a & \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} & \mathsf{stk} \end{array}$$

normal form reduction

$$\Omega_{\mathcal{R}}\ f(f(i(i(a)),i(a)),a) \quad \longmapsto^{\mathsf{stk}}_{\rho\hat{\delta}\gamma} \quad i(i(a)) \wr a$$

This latter reduction expresses well the non-confluent reductions of the term $f(f(i(i(a)),i(a)),a)$ w.r.t. the TRS $\mathcal{R}$ since the result $i(i(a)) \wr a$ represents the two normal forms.

4 Conclusions

We have studied the confluence and the expressive power of a rewriting calculus featuring left-distributivity of the application over the structure, whereas only right-distributivity was available in former versions. The confluence of the calculus, which is endangered by careless distributivity of one operator over another, has been recovered using a *call-by-value* reduction, and is proved using the usual parallel reduction technique.

Since, in the rewriting calculus, a structure of ρ-rules can be seen as a naive encoding of a term rewrite system then the right-distributivity rule describes the application of each rewrite rule in the structure to the argument. Moreover, structures can be also used to denote the sets of results obtained as result of such an application and the left-distributivity describes the application of a given rule (or structure of rules) to many distinct arguments in parallel. Thus, we can encode the simultaneous exploration of many reduction paths in a term.

Using the left-distributivity together with some earlier techniques, we obtain a better handling of matching failures, and we are able to faithfully encode the behavior of any term rewriting system, even non-confluent. This allows for many interesting theoretical developments, such as the computation of all the normal forms of a given term, which is needed, for example, for the completion of a term rewriting system.

The extension to general term rewriting systems is considered as the next step of this work. A major difficulty when dealing with matching modulo some congruence consists in the multiplicity of solutions and since these solutions cannot be ordered in any natural way, the structure operator should be then considered as associative, commutative and idempotent. Moreover the notion of definitive failures should be adapted to the considered matching theories and a call-by-value strategy should be enforced to prevent matching against uninstanciated terms and thus to avoid loosing matching solutions.

Related Work.

V. van Oostrom has widely studied the confluence of a λ-calculus with patterns [15], but which does not feature structures. Our encoding of TRS shares some similarities with the one presented by S. Byun *et al.* [3] that describes an untyped encoding of every strongly separable orthogonal TRS into λ-calculus. However, they need some really strong assumptions on the confluence of the original system.

Acknowledgement

We would like to thank Claude Kirchner for the useful interactions we had on the topics of this paper and to Germain Faure for his detailed and very useful comments on a preliminary version of this work.

References

[1] H. Ait-Kaci, A. Podelski, and G. Smolka. A feature constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1-2):263–283, 1994.

[2] P. Borovansky, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 2(285):155–185, 2002.

[3] S. Byun, R. Kennaway, V. van Oostrom, and F. de Vries. Separability and translatability of sequential term rewrite systems into the lambda calculus. Technical Report tr-2001-16, University of Leicester, 2001.

[4] H. Cirstea, C. Houtmann, and B. Wack. Distributive rho-calculus. Technical report, INRIA Lorraine, 2006. Available at http://hal.inria.fr.

[5] H. Cirstea and C. 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.

[6] H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In A. Middeldorp, editor, *Proceedings of RTA'2001*, Lecture Notes in Computer Science, Utrecht (The Netherlands), May 2001. Springer-Verlag.

[7] H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. Post-procededings of TYPES, 2003.

[8] K. Futatsugi and A. Nakagawa. An Overview of Cafe Project. In *Proc. of CafeOBJ Workshop*, 1996.

[9] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[10] G. Huet. *Resolution d'Equations dans les Languages d'Ordre 1,2,...,ω*. These de Doctorat D'Etat, Universite Paris VII, 1976.

[11] D. E. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.

[12] M. Takahashi. Parallel reduction in lambda-calculus. *Information and Computation*, 118(1):120–127, 1995.

[13] The Maude Team. The Maude Home Page, 2003. http://maude.cs.uiuc.edu/.

[14] A. van Deursen. An Overview of ASF+SDF. In *Language Prototyping*, pages 1–31. World Scientific, 1996.

[15] V. van Oostrom. Lambda Calculus with Patterns. Technical Report IR-228, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, 1990.

[16] B. Wack. *Typage et déduction dans le calcul de réécriture*. Thèse de doctorat, Université Henri Poincaré - Nancy I, Oct. 2005.