The rewriting calculus — Part II

HORATIU CIRSTEA, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France.*
E-mail: Horatiu.Cirstea@loria.fr

CLAUDE KIRCHNER, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France.*
E-mail: Claude.Kirchner@loria.fr

Abstract

The ρ-calculus integrates in a uniform and simple setting first-order rewriting, λ-calculus and non-deterministic computations. Its abstraction mechanism is based on the rewrite rule formation and its main evaluation rule is based on matching modulo a theory T.

We have seen in the first part of this work the motivations, definitions and basic properties of the ρ-calculus. This second part is first devoted to the use of an extension of the ρ-calculus for encoding a (conditional) rewrite relation. This extension is based on the *first* operator whose purpose is to detect rule application failure. It allows us to express recursively rule application and therefore to encode strategy based rewriting processes. We then use this extended calculus to give an operational semantics to ELAN programs.

We conclude with an overview of ongoing and future works on ρ-calculus.

Keywords: rewriting, strategy, non-determinism, matching, rewriting-calculus, lambda-calculus, rule based language.

1 Introduction

This is the second part of the rewriting calculus description, study and applications. In all the paper, we refer to the first part of this work as *Part I*.

As we have seen in *Part I*, we can encode in ρ-calculus the representation of a finite derivation. But we need more since we want to be able the represent also in the calculus the generic search for normalization derivations, when they exist. More generally, we want to have a formal representation of rewriting strategies like the ones used in ELAN [Pro01].

To this end we extend the calculus with a *first* operator whose purpose is to detect rule application failure. This extension allows us to express recursively rule application and therefore to encode strategy based rewriting processes.

We then extend the ρ-encoding of conditional rewriting to more complicated rules like the conditional rewrite rules with local assignments from the ELAN language. The non-determinism that in ELAN is handled mainly by two basic strategy operators is represented in the ρ-calculus by means of sets. We show finally how the ρ-calculus provides a semantics to ELAN programs.

This paper is structured as follows. In Section 2 we extend the basic ρ-calculus with a new operator and define term traversal and fixed-point operators using the existing ρ-operators.

The encoding of non-conditional and conditional term rewriting by using the ρ-operators defined in Section 2 is presented in Section 3. The calculus is finally used in Section 4 in order to give an operational semantics to the rules used in the ELAN language.

We conclude by providing some of the research directions that are of main interest in the development of this formalism and in the context of ELAN, and more generally of rewrite based languages as ASF+SDF [Kli93], ML [Mil84], Maude [CELM96], Stratego [Vis99] or CafeOBJ [FN97].

2 Recursion and term traversal operators

In *Part I* we have shown that for any reduction in a rewrite theory there exists a corresponding reduction in the ρ-calculus: if the term u reduces to the term v in a rewrite theory $\mathcal{R}$ we can build a ρ-term $\xi_{\mathcal{R}}(u)$ that reduces to the term $\{v\}$. The method used for constructing the term $\xi_{\mathcal{R}}(u)$ depends on all the reduction steps from u to v in the theory $\mathcal{R}$: $\xi_{\mathcal{R}}(u)$ is a representation in the ρ-calculus of the derivation trace. We want to go further on and to give a method for constructing a term $\xi_{\mathcal{R}}(u)$ without knowing a priori the derivation from u to v. Hence we want to answer to the following question:

Given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u, if u reduces to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to a set containing the term v?

This means that we wish to describe in the ρ-calculus reduction strategies and, mainly, normalization strategies. This will allow us to get, in particular, a natural encoding of normal conditional term rewriting. Therefore, we want to answer the more specific question:

Given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u if u normalizes to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to a set containing the term v?

The definition of normalization strategies is in general done at the *meta-level* while the ρ-calculus allows us to represent such derivations at the *object level*. We have shown in *Part I* that the $\rho_{\emptyset}$-calculus contains the λ-calculus and thus, any computable function as the normalization one is expressible in the formalism. What we bring here, because of the matching power and of the use of non-determinism, is an increased ease in the expression of such functions together with their expression in a uniform formalism combining standard rewrite techniques and higher-order behaviors.

When computing the normal form of a term u w.r.t. a rewrite system $\mathcal{R}$, the rewrite rules are applied *repeatedly* at *any position* of a term u until no rule from $\mathcal{R}$ is *applicable*. Hence, the ingredients needed for defining such a strategy are:

- an iteration operator that applies *repeatedly* a set of rewrite rules,
- a term traversal operator that applies a rewrite rule at *any position* of a term,
- an operator testing if a set of rewrite rules is *applicable* to a term.

In what follows we describe how the operators with the above functionalities can be defined in the ρ-calculus. We start with some auxiliary operators and afterwards, we introduce the ρ-operators that correspond to the functionalities listed above.

2.1 Some auxiliary operators

First, we define three auxiliary operators that will be used in the next sections. These operators are just aliases used to define more complex ρ-terms and are used for giving more compact and clear definitions for the recursion operators.

The first of these operators is the *identity* (denoted id) that applied to any ρ-term t evaluates to the singleton containing this term, that is $[id](t) \longrightarrow_\rho \{t\}$. The ρ-term id is nothing else but the rewrite rule $x \to x$:

$$id \triangleq x \to x.$$

In a similar way we can define the strategy $fail$ which always fails, (*i.e.* applied to any term, leads to $\emptyset$):

$$fail \triangleq x \to \emptyset.$$

The third one is the binary operator ";" that represents the sequential application of two ρ-terms. A ρ-term of the form $[u; v](t)$ represents the application of the term v to the result of the application of u to t. Therefore, we define the operator ";" by:

$$u; v \triangleq x \to [v]([u](x)).$$

In the following sections we generally employ the abbreviations of these operators and not their expanded form but we sometimes show the corresponding reductions.

2.2 The first operator

We introduce now a new operator, similar to the THEN operator for combining tactics and already present in LCF [GMW79]. Its role is to select between its arguments the first one that applied to a given ρ-term does not evaluate to $\emptyset$. If all the arguments evaluate to $\emptyset$ then the final result of the evaluation is $\emptyset$. The evaluation rules describing the $first$ operator and the auxiliary operator $\langle \_, \ldots, \_ \rangle$ are presented in Figure 1. We do not know currently how to express these operators in the basic ρ-calculus and we conjecture that this is not possible.

For simplicity, we considered that the operators $first$ and $\langle\rangle$ are of variable arity but similar binary operators can be used instead.

The application of a ρ-term $first(s_1, \ldots, s_n)$ to a term t returns the result of the first "successful" application of one of its arguments to the term t. Hence, if $[s_i](t)$ evaluates to $\emptyset$ for $i = 1, \ldots, k-1$, and $[s_k](t)$ does not evaluate to $\emptyset$, then $[first(s_1, \ldots, s_n)](t)$ evaluates to the same term as the term $[s_k](t)$.

If the evaluation of the terms $[s_i](t)$, $i = 1, \ldots, k-1$, leads to $\emptyset$ and the evaluation of $[s_k](t)$ does not terminate then the evaluation of the term $[first(s_1, \ldots, s_n)](t)$ does not terminate.

DEFINITION 2.1
The set of ρ^{1st}-terms extends the set $\varrho(\mathcal{F}, \mathcal{X})$ of basic ρ-terms, with the following two rules:

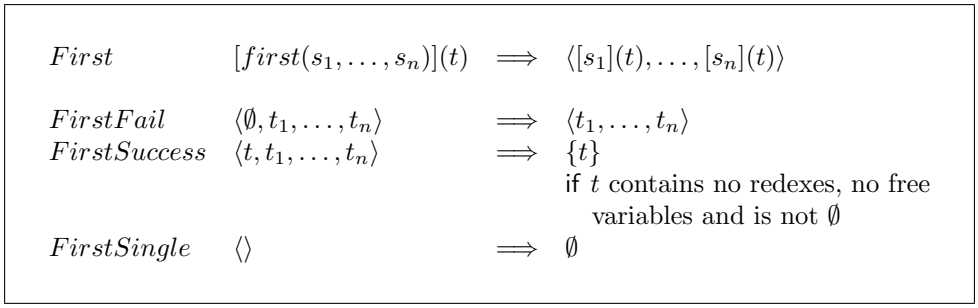

FIG. 1. The $first$ operator

- if $t_1,\ldots,t_n$ are ρ-terms then $first(t_1,\ldots,t_n)$ is a ρ-term,
- if $t_1,\ldots,t_n$ are ρ-terms then $\langle t_1,\ldots,t_n\rangle$ is a ρ-term.

This set of terms is denoted by $\varrho^{1st}(\mathcal{F},\mathcal{X})$.

We define now the ρ^{1st}_T-calculus by considering the new operators and the corresponding evaluation rules presented in Figure 1:

DEFINITION 2.2
Given a set $\mathcal{F}$ of function symbols, a set $\mathcal{X}$ of variables, a theory T on $\varrho^{1st}(\mathcal{F},\mathcal{X})$ terms having a decidable matching problem, we call ρ^{1st}_T-calculus a calculus defined by:

- a non-empty subset $\varrho^{1st}_{-}(\mathcal{F},\mathcal{X})$ of the $\varrho^{1st}(\mathcal{F},\mathcal{X})$ terms,
- the (higher-order) substitution application to terms as defined in *Part I*,
- a theory T,
- the set of evaluation rules $\mathcal{E}_{\rho^{1st}}$: $Fire$, $Cong$, $CongFail$, $Distrib$, $Batch$, $Switch_L$, $Switch_R$, $OpOnSet$, $Flat$, $First$, $FirstFail$, $FirstSuccess$, $FirstSingle$,
- an evaluation strategy $\mathcal{S}$ that guides the application of the evaluation rules.

In what follows we consider the ρ^{1st}-calculus, *i.e.* the ρ^{1st}_T-calculus with a syntactic matching and whose rewrite rules are restricted to be of the form $u \to v$ where u is a first-order term.

The following examples present the evaluation of some ρ^{1st}-terms containing the operators of the extended calculus.

EXAMPLE 2.3
The non-deterministic application of one of the rules $a \to b$, $a \to c$, $a \to d$ to the term a is represented in the ρ-calculus by the application $[\{a \to b, a \to c, a \to d\}](a)$. This last ρ-term is reduced to the term $\{b,c,d\}$ which represents a non-deterministic choice among the three terms. If we want to apply the above rules in a deterministic way and in the specified order, we use the ρ-term $[first(a \to b, a \to c, a \to d)](a)$ with, for example, the reduction:

$$\begin{array}{ll} & [first(a \to b, a \to c, a \to d)](a) \\ \longrightarrow_{First} & \langle [a \to b](a), [a \to c](a), [a \to d](a)\rangle \\ \longrightarrow_{Fire} & \langle \{b\}, [a \to c](a), [a \to d](a)\rangle \end{array}$$

$$\begin{array}{ll} \longrightarrow_{FirstSuccess} & \{\{b\}\} \\ \longrightarrow_{Flat} & \{b\} \end{array}$$

We can notice that even if all the rewrite rules can be applied successfully (*i.e.* no empty set) to the term a, the final result is given by the first tried rewrite rule.

Example 2.4
We consider now the case where some of the rules given in argument to $first$ lead to an empty set result:

$$\begin{array}{ll} & [first(a \to b, b \to c, a \to d)](b) \\ \longrightarrow_{First} & \langle [a \to b](b), [b \to c](b), [a \to d](b) \rangle \\ \longrightarrow_{Fire} & \langle \emptyset, [b \to c](b), [a \to d](b) \rangle \\ \longrightarrow_{FirstFail} & \langle [b \to c](b), [a \to d](b) \rangle \\ \longrightarrow_{Fire} & \langle \{c\}, [a \to d](b) \rangle \\ \longrightarrow_{FirstSuccess} & \{\{c\}\} \\ \longrightarrow_{Flat} & \{c\} \end{array}$$

Example 2.5
If none of the rules given in argument to $first$ is applied successfully, the result is obviously the empty set:

$$\begin{array}{ll} & [first(a \to b, a \to c, a \to d)](b) \\ \longrightarrow_{First} & \langle [a \to b](b), [a \to c](b), [a \to d](b) \rangle \\ \stackrel{*}{\longrightarrow}_{Fire} & \langle \emptyset, \emptyset, \emptyset \rangle \\ \stackrel{*}{\longrightarrow}_{FirstFail} & \langle \rangle \\ \longrightarrow_{FirstSingle} & \emptyset \end{array}$$

The operator $first$ does not test explicitly the applicability of a term (rule) to another term but allows us to recover from a failure and continue the evaluation. For example, we can define a term

$$try(s) \triangleq first(s, id)$$

that applied to the term t evaluates to the result of $[s](t)$, if $[s](t)$ does not evaluate to $\emptyset$ and to $\{t\}$, if $[s](t)$ evaluates to $\emptyset$.

2.3 Term traversal operators

Let us now define operators that apply a ρ-term at some position of another ρ-term. The first step is the definition of two operators that push the application of a ρ-term one level deeper on another ρ-term. This is already possible in the ρ-calculus due to the rule $Cong$ but we want to define a generic operator that applies a ρ-term r to the sub-terms u_i, $i = 1 \dots n$, of a term of the form $F(u_1, \dots, u_n)$ independently on the head symbol F.

To this end, we define two term traversal operators, $\Phi(r)$ and $\Psi(r)$, whose behavior is described by the rules in Figure 2. These operators are inspired by the operators of the *System S* described in [VeAB98].

The application of the ρ-term $\Phi(r)$ to a term $t = f(u_1, \dots, u_n)$ results in the successful application of the term r to one of the terms u_i. More precisely, r is

$$
\begin{array}{lll}
TraverseSeq & [\Phi(r)](f(u_1,\ldots,u_n)) \implies & \\
 & \langle\{f([r](u_1),\ldots,u_n)\},\ldots,\{f(u_1,\ldots,[r](u_n))\}\rangle & \\
 & & \\
TraversePar & [\Psi(r)](f(u_1,\ldots,u_n)) \implies & \{f([r](u_1),\ldots,[r](u_n))\}
\end{array}
$$

FIG. 2. The term traversal operators of the ρ_T-calculus

applied to the first u_i, $i = 1,\ldots,n$ such that $[r](u_i)$ does not evaluate to the empty set. If there *exists no* such u_i and in particular, if t is a function with no arguments (t is a constant), then the term $[\Phi(r)](t)$ reduces to the empty set:

$$[\Phi(r)](c) \longrightarrow_{TraverseSeq} \langle\{\}\rangle \longrightarrow_{FirstFail} \langle\rangle \longrightarrow_{FirstSingle} \emptyset$$

When the ρ-term $\Psi(r)$ is applied to a term $t = f(u_1,\ldots,u_n)$ the term r is applied to all the arguments u_i, $i = 1,\ldots,n$ if *for all* i, $[r](u_i)$ does not evaluate to $\emptyset$. If there exists an u_i such that $[r](u_i)$ reduces to $\emptyset$, then the result is the empty set. If we apply $\Psi(r)$ to a constant c, since there are no sub-terms the term $[\Psi(r)](c)$ reduces to $\{c\}$:

$$[\Psi(r)](c) \longrightarrow_{TraversePar} \{c\}$$

If we consider a ρ-calculus with a finite signature $\mathcal{F}$ and if we denote by $\mathcal{F}_0 = \{c_1,\ldots,c_n\}$ the set of constant function symbols and by $\mathcal{F}_+ = \{f_1,\ldots,f_m\}$ the set of function symbols with arity at least one, the two term traversal operators can be expressed in the ρ-calculus by some appropriate ρ-terms.

If the following two definitions are considered

$$\Phi'(r) \triangleq first(f_1(r,id,\ldots,id),\ldots,f_1(id,\ldots,id,r),\ldots, f_m(r,id,\ldots,id),\ldots,f_m(id,\ldots,id,r))$$

$$\Psi(r) \triangleq \{c_1,\ldots,c_n,f_1(r,\ldots,r),\ldots,f_m(r,\ldots,r)\}$$

with $c_i \in \mathcal{F}_0$, $i = 1,\ldots,n$, and $f_j \in \mathcal{F}_+$, $j = 1,\ldots,m$, we obtain the following two reductions,

$$
\begin{array}{ll}
 & [\Phi'(r)](f_k(u_1,\ldots,u_p)) \\
\triangleq & [first(f_1(r,id,\ldots,id),\ldots,f_m(id,\ldots,id,r))](f_k(u_1,\ldots,u_p)) \\
\longrightarrow_{First} & \langle[f_1(r,id,\ldots,id)](f_k(u_1,\ldots,u_p)),\ldots,[f_m(id,\ldots,id,r)](f_k(u_1,\ldots,u_p))\rangle \\
\xrightarrow{*}_{Cong} & \langle\emptyset,\ldots,\emptyset,\{f_k([r](u_1),\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle \\
\xrightarrow{*}_{FirstFail} & \langle\{f_k([r](u_1),\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle
\end{array}
$$

and

$$
\begin{array}{ll}
 & [\Psi(r)](f_k(u_1,\ldots,u_p)) \\
\triangleq & [\{c_1,\ldots,c_n,f_1(r,\ldots,r),\ldots,f_m(r,\ldots,r)\}](f_k(u_1,\ldots,u_p)) \\
\longrightarrow_{Distrib} & \{[c_1](f_k(u_1,\ldots,u_p)),\ldots\ldots,[f_m(r,\ldots,r)](f_k(u_1,\ldots,u_p))\} \\
\xrightarrow{*}_{Cong} & \{\emptyset,\ldots,\emptyset,\{f_k([r](u_1),\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\} \\
\xrightarrow{*}_{Flat} & \{f_k([r](u_1),\ldots,[r](u_p))\}
\end{array}
$$

The operator Φ' does not correspond exactly to the definition from the Figure 2 but, as we have just seen above, a similar result is obtained when applying the terms $\Phi(r)$ and $\Phi'(r)$ to a term $f_k(u_1, \ldots, u_p)$.

LEMMA 2.6
The term traversal operators Φ and Ψ can be expressed in the ρ^{1st}-calculus.

PROOF. If we consider $t = f_k(u_1, \ldots, u_p)$ and if for all $i = 1, \ldots, p$ we have the reductions $[r](u_i) \xrightarrow{*}_\rho \emptyset$ then, according to the evaluation rules describing the behavior of $\Phi(r)$, we obtain:

$$\begin{array}{ll}
 & [\Phi(r)](f_k(u_1, \ldots, u_p)) \\
\longrightarrow_{TraverseSeq} & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\} \rangle \\
\xrightarrow{*}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\} \rangle \\
\xrightarrow{*}_{OpOnSet} & \langle \{\emptyset\}, \ldots, \{\emptyset\} \rangle \\
\xrightarrow{*}_{Flat} & \langle \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_{FirstFail} & \langle \rangle \\
\longrightarrow_{FirstSingle} & \emptyset
\end{array}$$

Otherwise, if there exists an l such that $[r](u_i) \xrightarrow{*}_\rho \emptyset$, $i = 1, \ldots, l-1$ and $[r](u_l) \xrightarrow{*}_\rho v_l \downarrow$, with $v_l \downarrow$ a ground term containing no redex, the following reduction is obtained:

$$\begin{array}{ll}
 & [\Phi(r)](f_k(u_1, \ldots, u_p)) \\
\longrightarrow_{TraverseSeq} & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\} \rangle \\
\xrightarrow{*}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, v_l \downarrow, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\} \rangle \\
\xrightarrow{*}_{OpOnSet} & \langle \emptyset, \ldots, \emptyset, \{f_k(u_1, \ldots, v_l \downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_{FirstFail} & \langle \{f_k(u_1, \ldots, v_l \downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset \rangle
\end{array}$$

Depending on the evaluation strategy, the terms following $f_k(u_1, \ldots, v_l \downarrow, \ldots, u_p)$ can be reduced or not to the empty set and we have chosen here the former alternative for a more compact representation.

Now, if we consider the definition of $\Phi'(r)$ and if for all $i = 1, \ldots, p$ we have $[r](u_i) \xrightarrow{*}_\rho \emptyset$ then, we obtain:

$$\begin{array}{ll}
 & [\Phi'(r)](f_k(u_1, \ldots, u_p)) \\
\xrightarrow{*}_\rho & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\}, \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\}, \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_{OpOnSet} & \langle \{\emptyset\}, \ldots, \{\emptyset\}, \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_{Flat} & \langle \emptyset, \ldots, \emptyset, \ldots, \emptyset \rangle \\
\xrightarrow{*}_{FirstFail} & \langle \rangle \\
\longrightarrow_{FirstSingle} & \emptyset
\end{array}$$

For the same term $[\Phi'(r)](f_k(u_1, \ldots, u_p))$, if it exists an l such that $[r](u_i) \xrightarrow{*}_\rho \emptyset$, $i = 1, \ldots, l-1$ and $[r](u_l) \xrightarrow{*}_\rho v_l \downarrow$, with $v_l \downarrow$ a ground term containing no redex, the following reduction is obtained:

$$[\Phi'(r)](f_k(u_1, \ldots, u_p))$$

$$\begin{array}{ll}
\xrightarrow{*}_{\rho} & \langle\{f_k([r](u_1),\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle \\
\xrightarrow{*}_{\rho} & \langle\{f_k(\emptyset,\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,v_l\downarrow,\ldots,u_p)\},\emptyset,\ldots,\emptyset\rangle \\
\xrightarrow{*}_{OpOnSet} & \langle\{\emptyset\},\ldots,\{\emptyset\},\{f_k(u_1,\ldots,v_l\downarrow,\ldots,u_p)\},\emptyset,\ldots,\emptyset\rangle \\
\xrightarrow{*}_{Flat} & \langle\emptyset,\ldots,\emptyset,\{f_k(u_1,\ldots,v_l\downarrow,\ldots,u_p)\},\emptyset,\ldots,\emptyset\rangle \\
\xrightarrow{*}_{FirstFail} & \langle\{f_k(u_1,\ldots,v_l\downarrow,\ldots,u_p)\},\emptyset,\ldots,\emptyset\rangle
\end{array}$$

We can notice that the results of the reductions for the application of a term r to the arguments of a term $f_k(u_1,\ldots,u_p)$ by using the two operators, Φ and Φ', are identical. If the terms u_i, $i=1\ldots p$, are ground terms containing no redex then, the final result of the two reductions in the case without failure is $\{f_k(u_1,\ldots,v_l\downarrow,\ldots,u_p)\}$.

When the operators are applied to a constant $c_k\in\mathcal{F}_0$ we obtain:

$$[\Phi'(r)](c_k)\xrightarrow{*}_{\rho}\langle\rangle\longrightarrow_{\rho}\emptyset,$$

$$[\Psi(r)](c_k)\xrightarrow{*}_{\rho}\{c_k\}.$$

□

2.4 Iterators

The definition of the evaluation (normalization) strategies as, for example, *top-down* or *bottom-up*, is based on the application of one term to the top position or to the deepest positions of another term.

For the moment, we have the possibility of applying a ρ-term r either to one or all the arguments u_i of a ρ-term $t=f(u_1,\ldots,u_n)$, or to the sub-terms of t at an explicitly specified depth. But the depth of a term is not known *a priori* and thus, we cannot apply a term r to the deepest positions of a term t. If we want to apply the term r to the sub-terms at the maximum depth of a term t we must define a recursive operator which reiterates the application of the $\Phi(r)$ and $\Psi(r)$ terms and thus, pushes the application deeper into terms.

We start by presenting the ρ-term used for describing recursive applications in the ρ-calculus. Starting from the fixed-point combinators of the λ-calculus, we define a ρ-term which recursively applies a given ρ-term. We use the classical fixed-point combinator of the λ-calculus ([Bar84]), $\Theta_\lambda=(A_\lambda\ A_\lambda)$ where

$$A_\lambda=\lambda xy.y(xxy)$$

and Θ_λ is called the Turing fixed-point combinator ([Tur37]).

This term corresponds in the ρ-calculus to the ρ-term $\Theta=[A](A)$ with

$$A=x\to(y\to[y]([[x](x)](y))).$$

In λ-calculus, for any λ-term G we have the reduction

$$\Theta_\lambda\ G\xrightarrow{*}_{\beta}G(\Theta_\lambda\ G).$$

In ρ-calculus, we have a similar reduction

$$[\Theta](G)\xrightarrow{*}_{\rho}\{[G]([\Theta](G))\}\qquad(Fixed\ Point)$$

as this can be checked as follows:

$$
\begin{array}{ll}
 & [\Theta](G) \triangleq [[A](A)](G) \triangleq [[x \to (y \to [y]([[x](x)](y)))](A)](G) \\
\longrightarrow_{Fire} & [\{y \to [y]([[A](A)](y))\}](G) \\
\longrightarrow_{Distrib} & \{[y \to [y]([[A](A)](y))](G)\} \\
\longrightarrow_{Fire} & \{\{[G]([[A](A)](G))\}\} \\
\longrightarrow_{Flat} & \{[G]([[A](A)](G))\} \\
\triangleq & \{[G]([\Theta](G))\}
\end{array}
$$

We have obtained the desired result but the last application of the rule $Fire$ in the above reduction can be replaced by a reduction in the sub-term $[[A](A)](y)$. We can thus reduce $[[A](A)](y) \triangleq [[x' \to (y' \to [y']([[x'](x')](y')))](A)](y)$ to the term $\{[y]([[A](A)](y))\} \triangleq \{[y]([\Theta](y))\}$. We therefore obtain the following derivation:

$$
\begin{array}{ll}
 & [\Theta](G) \\
\xrightarrow{*}_{\rho} & \{[y \to [y]([[A](A)](y))](G)\} \triangleq \{[y \to [y]([\Theta](y))](G)\} \\
\xrightarrow{*}_{\rho} & \{[y \to [y](\{[y]([\Theta](y))\})](G)\} \\
\xrightarrow{*}_{\rho} & \{[y \to [y]([y]([\Theta](y)))](G)\} \\
\xrightarrow{*}_{\rho} & \ldots
\end{array}
$$

which does not terminate if the same redex $[\Theta](y)$ is always selected for reduction.

In an operational approach we do not want the new constructions to lead to non-terminating reductions. Since the ρ-term $[\Theta](G)$ can obviously lead to infinite reductions, a strategy should be used in order to obtain termination and thus the desired behavior.

We should thus use a strategy which applies the evaluation rules to a sub-term of the form $[\Theta](G)$ only when no other reduction is possible. From an operational point of view, this strategy is rather difficult to implement and obviously not very efficient in a calculus where the Θ term is represented by its extended form and thus, more difficult to identify. If Θ is considered as an independent ρ-term with the behavior described by an evaluation rule corresponding to the reduction ($Fixed\ Point$), the strategy suggested previously could be easily implemented.

A strategy satisfying the termination condition and easier to implement could initially apply the evaluation rules at the top positions of the terms and only when no evaluation rule can be applied at the top position, reduce the sub-terms at deeper positions. In what follows we will generally use this *outermost* strategy. It is clear that such a strategy prevents only the infinite reductions due to the operator Θ, but it cannot ensure the termination of the untyped ρ-calculus.

As we mentioned previously, the main goal of this section is the representation of normalization strategies by ρ-terms and thus, we want to describe the application of a term r to all the positions of another term t. Therefore, we must define the appropriate term G that propagates the application of a ρ-term in the sub-terms of another ρ-term.

2.4.1 Multiple applications

First, we want to define the operators $BottomUp$ and $TopDown$ describing the application of a term r to all the sub-terms of a term t starting with the deepest positions of t and respectively with the top position of t. We want thus to find a term which

recursively applies the term r to all the sub-terms of t and afterwards at the top position of the result term and another term which initially applies the term r at the top position of the term t and then to the sub-terms of the result term. The term r must be applied to the sub-terms only if this application does not lead to a failure.

We propose first two "naive" definitions for the former operator and we comment the encountered problems. We analyze the obtained reductions and we define afterwards the operators describing the desired behavior.

The first natural possibility is to define the ρ-term

$$G_{sds}(r) \triangleq f \to (x \to [\Psi(f); r](x))$$

Let us consider the ρ-term SDS (for *SpreadDownSimple*),

$$SDS(r) \triangleq [\Theta](G_{sds}(r))$$

and its application to the term $t = f(t_1, \ldots, t_n)$. Then, the following derivation is obtained:

$$\begin{array}{ll}
 & [SDS(r)](t) \triangleq [[\Theta](G_{sds}(r))](t) \\
\xrightarrow{*}_{\rho} & \{[[G_{sds}(r)]([\Theta](G_{sds}(r)))](t)\} \\
\triangleq & \{[[G_{sds}(r)](SDS(r))](t)\} \\
\triangleq & \{[[f \to (x \to [\Psi(f); r](x))](SDS(r))](t)\} \\
\xrightarrow{*}_{\rho} & \{[\{x \to [\Psi(SDS(r)); r](x)\}](t)\} \\
\xrightarrow{*}_{\rho} & \{[\Psi(SDS(r)); r](f(t_1, \ldots, t_n))\} \\
\xrightarrow{*}_{\rho} & \{[r]([\Psi(SDS(r))](f(t_1, \ldots, t_n)))\} \\
\xrightarrow{*}_{\rho} & \{[r](f([SDS(r)](t_1), \ldots, [SDS(r)](t_n)))\}
\end{array}$$

As we can see from this derivation, the term $SDS(r)$ is recursively applied to the sub-terms of the initial term and the term r is applied at the top position of the result. If one of the applications of the term r leads to a failure, then this failure is propagated and the empty set is obtained as the result of the derivation.

When using a confluent strategy, as the ones presented in *Part I*, the derivation presented above is possible only if the term $G_{sds}(r)$ cannot be reduced to a set with more than one element. This condition is obviously not respected if r is a set with more than one element since, for example, $G_{sds}(\{a, b\}) \xrightarrow{*}_{\rho} \{G_{sds}(a), G_{sds}(b)\}$. We want to prevent the evaluation of the term $G_{sds}(r)$ to a set with more than one element even when r does not satisfy this condition and therefore, we define the term

$$G_{sd}(r) \triangleq f \to (x \to \langle [\Psi(f); r](x) \rangle)$$

and respectively SD (for *SpreadDown*),

$$SD(r) \triangleq [\Theta](G_{sd}(r)).$$

If $r = \{a, b\}$ then, the term $G_{sd}(r) = G_{sd}(\{a, b\})$ is not reduced to the term $\{G_{sd}(a), G_{sd}(b)\}$ as it was the case for $G_{sds}(r)$ but

$$\begin{array}{ll}
 & G_{sd}(r) \triangleq f \to (x \to \langle [\Psi(f); \{a, b\}](x) \rangle) \\
\xrightarrow{*}_{\rho} & f \to (x \to \langle [\{a, b\}]([\Psi(f)](x)) \rangle) \\
\xrightarrow{*}_{Distrib} & f \to (x \to \langle \{[a]([\Psi(f)](x)), [b]([\Psi(f)](x))\} \rangle)
\end{array}$$

In this last term, the first argument of the operator $\langle\rangle$ contains the free variable x and thus, it cannot be reduced by using the evaluation rule $FirstSuccess$.

Since this last term is not a set, the propagation of the set symbols is not performed in the case of the operator G_{sd} and we can reduce the term $[\Theta](G_{sd}(r))$ to $\{[G_{sd}(r)]([\Theta](G_{sd}(r)))\}$. Consequently, we obtain the reduction:

$$\begin{array}{ll} & [SD(r)](t) \triangleq [[\Theta](G_{sd}(r))](t) \\ \xrightarrow{*}_{\rho} & \{[[G_{sd}(r)]([\Theta](G_{sd}(r)))](t)\} \\ \triangleq & \{[[G_{sd}(r)](SD(r))](t)\} \\ \triangleq & \{[[f \to (x \to \langle[\Psi(f); r](x)\rangle)](SD(r))](t)\} \\ \xrightarrow{*}_{\rho} & \{[\{x \to \langle[\Psi(SD(r)); r](x)\rangle\}](t)\} \\ \xrightarrow{*}_{\rho} & \{\langle[\Psi(SD(r)); r](f(t_1, \ldots, t_n))\rangle\} \\ \xrightarrow{*}_{\rho} & \{\langle[r](f([SD(r)](t_1), \ldots, [SD(r)](t_n)))\rangle\} \end{array}$$

Example 2.7

If we use a strategy which initially applies the evaluation rules at the top positions of terms then, the following derivation is obtained:

$$\begin{array}{ll} & [SD(\{a \to b, id\})](g(a, f(a))) \\ \xrightarrow{*}_{\rho} & \{\langle[\{a \to b, id\}](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\rangle\} \\ \longrightarrow_{Distrib} & \{\langle\{[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))), \\ & \quad [id](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\}\rangle\} \\ \longrightarrow_{Fire} & \{\langle\{\emptyset, [id](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\}\rangle\} \\ \longrightarrow_{Flat} & \{\langle\{g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\ \xrightarrow{*}_{\rho} & \{\langle\{g(\{\langle[\{a \to b, id\}](a)\rangle\}, [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\ \xrightarrow{*}_{\rho} & \{\langle\{g(\{\{b, a\}, [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\ \xrightarrow{*}_{\rho} & \{\langle\{g(\{b, a\}, \langle[\{a \to b, id\}](f([SD(\{a \to b, id\})](a))))\}\rangle\} \\ \xrightarrow{*}_{\rho} & \{\langle\{g(\{b, a\}, f(\{b, a\}))\}\rangle\} \\ \xrightarrow{*}_{\rho} & \{g(b, f(b)), g(a, f(b)), g(b, f(a)), g(a, f(a))\} \end{array}$$

We can notice that the application $[SD(r)](t)$ does not guarantee that the applications of the term r to the deepest sub-terms of t are the first ones to be reduced. For example, since we try to apply the evaluation rules at the top position, in the derivation of Example 2.7 we obtain, by applying the evaluation rule $Fire$,

$$[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))) \longrightarrow_{Fire} \emptyset$$

and not

$$[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))$$
$$\xrightarrow{*}_{\rho} [a \to b](g(\{b, a\}, \{f(\{b, a\})\})) \xrightarrow{*}_{\rho} \emptyset$$

as in an *innermost* reduction.

The disadvantage of the non-confluence in the case of the operator SDS was eliminated by using the operator $\langle\rangle$ in the definition of the operator SD, but we have not obtained yet the desired behavior for this type of iterator. In the evaluation of the term $[SD(r)](t)$, if one of the applications of the term r to a sub-term of t is evaluated to $\emptyset$ then, this failure is propagated and the empty set is obtained as the result of the reduction.

If we want to keep unchanged the sub-terms of t on which the application of the term r evaluates to $\emptyset$, we can use the term id either in the same way as in Example 2.7, or by defining the operator G_{bu}:

$$G_{bu}(r) \triangleq f \to (x \to [first(\Psi(f), id); first(r, id)](x))$$

In the same manner as for the previous cases we obtain the operator $BottomUp$:

$$BottomUp(r) \triangleq [\Theta](G_{bu}(r))$$

corresponding to the description presented at the beginning of this section.

LEMMA 2.8
The $BottomUp$ operator describing the application of a term to all the sub-terms of another term in a *bottom-up* manner can be expressed in the ρ^{1st}-calculus.

PROOF. We analyze the reductions of the application of a term $BottomUp(r)$ to a constant and to a functional term with several arguments. A complete proof is given in [Cir00]. □

A *top-down* like reduction is immediately obtained if we take the term

$$G_{td}(r) \triangleq f \to (x \to \langle [first(r, id); first(\Psi(f), id)](x) \rangle)$$

and we define the term

$$TopDown(r) \triangleq [\Theta](G_{td}(r)).$$

LEMMA 2.9
The $TopDown$ operator describing the application of a term to all the sub-terms of another term in a *top-down* manner can be expressed in the ρ^{1st}-calculus.

2.4.2 Singular applications

Using the term traversal operator Φ we can define similar ρ-terms that apply a specific term only at one position of a ρ-term in a *bottom-up* or *top-down* way. We will see that the operators built using the Φ operator are convenient for the construction of normalization operators.

The ρ-term used in the *bottom-up* case is

$$H_{bu}(r) \triangleq f \to (x \to [first(\Phi(f), r)](x))$$

and we define an operator that applies only once a ρ-term in a *bottom-up* way,

$$Once_{bu}(r) \triangleq [\Theta](H_{bu}(r)).$$

As for the previous operators, the term $[Once_{bu}(r)](t) \triangleq [[\Theta](H_{bu}(r))](t)$ can lead to an infinite reduction if an appropriate strategy is not employed. As for the $SpreadDown$ operator it is enough to apply the evaluation rules first to the top position and only if this is not possible, to deeper positions. We can state:

LEMMA 2.10
The $Once_{bu}$ operator describing the application of a term to a sub-term of another term in a *bottom-up* manner can be expressed in the ρ^{1st}-calculus.

Example 2.11
The application $[Once_{bu}(a \to b)](a)$ is reduced to $\{\langle[(a \to b)](a)\rangle\}$ and thus, to the term $\{b\}$.

The application of the rule $a \to b$ to the leftmost-innermost position of a term $g(a, f(a))$ is represented by the term $[Once_{bu}(a \to b)](g(a, f(a)))$ and the corresponding evaluation is presented below:

$$
\begin{array}{l}
\quad [Once_{bu}(a \to b)](g(a, f(a))) \\
\xrightarrow{*}_{\rho} \{\langle\langle g([Once_{bu}(a \to b)](a), f(a)), g(a, [Once_{bu}(a \to b)](f(a)))\rangle, [a \to b](g(a, f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\langle g(\{b\}, f(a)), g(a, [Once_{bu}(a \to b)](f(a)))\rangle, [a \to b](g(a, f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{g(b, f(a))\}, [a \to b](g(a, f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} \{g(b, f(a))\}
\end{array}
$$

If we want to define an operator that applies a specific term only at one position of a ρ-term in a *top-down* way we should use the ρ-term

$$H_{td}(r) \triangleq f \to (x \to [first(r, \Phi(f))](x))$$

and we obtain immediately the operator $Once_{td}$,

$$Once_{td}(r) \triangleq [\Theta](H_{td}(r)).$$

In the case of an application $[Once_{td}(r)](t)$, the application of the term r is first tried at the top position of t and in the case of a failure, r is applied deeper in the term t. As previously, we can state:

Lemma 2.12
The $Once_{td}$ operator describing the application of a term to a sub-term of another term in a *top-down* manner can be expressed in the ρ^{1st}-calculus.

2.5 Repetition and normalization operators

In the previous sections we have defined operators that describe the application of a term at some position of another term (*e.g.* $Once_{bu}$) and operators that allow us to recover from failing evaluations ($first$).

Now we want to define an operator that applies repeatedly a given strategy r to a ρ-term t. We call it *repeat* and its behavior can be described by the following evaluation rule:

$$Repeat \quad [repeat(r)](t) \quad \Longrightarrow \quad [repeat(r)]([r](t))$$

We use once again the fixed-point operator presented in the previous section and we define the ρ-term

$$I(r) \triangleq f \to (x \to [r; f](x))$$

that is used for describing a *repeat* operator,

$$repeat(r) \triangleq [\Theta](I(r)).$$

This approach has two obvious drawbacks. First, the termination of the evaluation is not guaranteed even when the strategy used for the previous operators is used.

When the strategy applies the evaluation rules first to the top position of an application $[u](v)$ and only afterwards to the right sub-term v and then to the left sub-term u, we do not obtain the desired result. When using this *rightmost-outermost* strategy, the following non-terminating derivation is obtained:

$$[repeat(r)](t) \xrightarrow{*}_\rho \{[repeat(r)]([r](t))\} \xrightarrow{*}_\rho \dots$$

$$\xrightarrow{*}_\rho \{[repeat(r)]([r]([r](\dots[r](t)\dots)))\} \xrightarrow{*}_\rho \dots$$

Second, when the evaluation terminates the result is always the empty set. If at some point in the evaluation the application of the term r is reduced to the empty set, then $\emptyset$ is strictly propagated and thus the term $[repeat(r)](t)$ is reduced to the empty set.

In order to overcome these two problems, we can define an operator called $repeat*$ with a behavior defined by the evaluation rules presented in Figure 3.

$Repeat*'$	$[repeat*(r)](t)$	$\Longrightarrow$	$[repeat*(r)]([r](t))$
			if $[r](t)$ is not reduced to $\emptyset$
$Repeat*''$	$[repeat*(r)](t)$	$\Longrightarrow$	t
			if $[r](t)$ is reduced to $\emptyset$

FIG. 3. The operator $repeat*$

Hence, we need an operator similar to the *repeat* one, that stores the last non-failing result and when no further application is possible returns this result. We modify the term $I(r)$ that becomes

$$J(r) \triangleq f \to (x \to [first(r; f,\ id)](x))$$

and we define, as before, the term

$$repeat*(r) \triangleq [\Theta](J(r))$$

We should not forget that we assume here that an application $[u](v)$ is reduced by applying the evaluation rules at the top position, then to its argument v and only afterwards to the term u. Once again, we get:

LEMMA 2.13
The operator $repeat*$ describing the repeated application of a term while the result is not $\emptyset$ can be expressed in the ρ^{1st}-calculus.

EXAMPLE 2.14
The repeated application of the rewrite rules $a \to b$ and $b \to c$ on the term a is represented by the term $[repeat*(\{a \to b, b \to c\})](a)$ that evaluates as follows:

$$\begin{aligned}
&[repeat*(\{a \to b, b \to c\})](a)\\
\xrightarrow{*}_\rho\ &\{\langle[repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](a)), [id](a)\rangle\}\\
\xrightarrow{*}_\rho\ &\{\langle[repeat*(\{a \to b, b \to c\})](\{b\}), [id](a)\rangle\}\\
\xrightarrow{*}_\rho\ &\{\langle\{\langle[repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](b)), [id](b)\rangle\}, [id](a)\rangle\}
\end{aligned}$$

$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\langle[repeat*(\{a \to b, b \to c\})](\{c\}), [id](b)\rangle\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\langle\{\langle[repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](c)), [id](c)\rangle\}, [id](b)\rangle\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\langle\{\langle[repeat*(\{a \to b, b \to c\})](\emptyset), \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\langle\{\langle\emptyset, \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\langle\{c\}, [id](b)\rangle\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{\langle\{\{c\}\}, [id](a)\rangle\}$
$\stackrel{*}{\longrightarrow}_\rho \{c\}$

Using the above operators it is easy to define some specific normalization strategies. For example, the *innermost* strategy is defined by

$$im(r) \triangleq repeat*(Once_{bu}(r))$$

and an *outermost* strategy is defined by

$$om(r) \triangleq repeat*(Once_{td}(r)).$$

Corollary 2.15
The operators im et om describing the *innermost* and *outermost* normalization can be expressed in the ρ^{1st}-calculus.

We have now all the ingredients needed for describing the normalization of a term t in a rewrite theory $\mathcal{R}$. The term $\xi_\mathcal{R}(u)$ described at the beginning of this section can be defined using the $im(\mathcal{R})$ or $om(\mathcal{R})$ operators and thus, we can represent the normalization of a term u w.r.t. a rewriting theory $\mathcal{R}$ by the ρ-terms

$$\xi_\mathcal{R}(u) \triangleq [im(\mathcal{R})](u)$$

or

$$\xi_\mathcal{R}(u) \triangleq [om(\mathcal{R})](u).$$

Example 2.16
If we denote by $\mathcal{R}$ the set of rewrite rules $\{a \to b, g(x, f(x)) \to x\}$, we represent by $[im(\mathcal{R})](g(a, f(a)))$ the leftmost-innermost normalization of the term $g(a, f(a))$ according to the set of rules $\mathcal{R}$ and the following derivation is obtained:

$$\begin{array}{ll}
 & [im(\mathcal{R})](g(a, f(a))) \\
\triangleq & [repeat*(Once_{bu}(\mathcal{R}))](g(a, f(a))) \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle[repeat*(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(a, f(a)))), [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle[repeat*(Once_{bu}(\mathcal{R}))](\{g(b, f(a))\}), [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle\{[repeat*(Once_{bu}(\mathcal{R}))](g(b, f(a)))\}, [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle\{\{\langle[repeat*(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b, f(a)))), \\
 & \quad [id](g(b, f(a)))\rangle\}\}, [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle\{\langle[repeat*(Once_{bu}(\mathcal{R}))](\{g(b, f(b))\}), \\
 & \quad [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle\{\langle\{\langle\{[repeat*(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b, f(b))))\}, \\
 & \quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\} \\
\stackrel{*}{\longrightarrow}_\rho & \{\langle\{\langle\{\langle\{[repeat*(Once_{bu}(\mathcal{R}))](\{b\})\}, \\
 & \quad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}
\end{array}$$

$$
\begin{aligned}
&\xrightarrow{*}_\rho \{\langle\{\langle\{\langle\{\langle[repeat*(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](b)), [id](b)\rangle,\\
&\qquad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\langle\{\langle\{\langle[repeat*(Once_{bu}(\mathcal{R}))](\emptyset), [id](b)\rangle,\\
&\qquad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\langle\{\langle\{\langle\emptyset, [id](b)\rangle,\\
&\qquad [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\langle\{\langle\{\langle\{\{b\}\}\rangle\}, [id](g(b, f(b)))\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\langle\{\langle\{\{\{b\}\}\rangle\}, [id](g(b, f(a)))\rangle\}, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\{\{b\}\}\rangle, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\langle\{\{b\}\rangle, [id](g(a, f(a)))\rangle\}\\
&\xrightarrow{*}_\rho \{\{b\}\}
\end{aligned}
$$

Given a term u, if the rewriting theory $\mathcal{R}$ is not confluent then, the result of the reduction of the term $[im(\mathcal{R})](u)$ is a set representing all the possible results of the reduction of the term u in the rewriting theory $\mathcal{R}$. Each of the elements of the result set represents the result of a reduction in the rewriting theory $\mathcal{R}$ for a given application order of the rewrite rules in $\mathcal{R}$.

Example 2.17
Let us consider the set $\mathcal{R} = \{a \to b, a \to c, g(x, x) \to x\}$ of non-confluent rewrite rules. The term $[im(\mathcal{R})](g(a, a))$ representing the *innermost* normalization of the term $g(a, a)$ according to the set of rewrite rules $\mathcal{R}$ is reduced to $\{b, g(c, b), g(b, c), c\}$. The term $[om(\mathcal{R})](g(a, a))$ representing the *outermost* normalization is reduced to $\{b, c\}$.

We have now all the ingredients necessary to describe in a concise way the normalization process induced by a rewrite theory. Of course, the standard properties of termination and confluence of the rewrite system will allow us to get uniqueness of the result. Our approach differs from this and we define this normalization even in the case where there is no unique normal form or where termination is not warranted. This is why in general we do not get termination or uniqueness of the normal form.

3 Using the ρ^{1st}-calculus

We have shown in *Part I* that a finite derivation in term rewriting can be mimicked as an appropriate ρ-term that indeed represents the trace of the reduction. It is often more interesting to *find* such a derivation.

3.1 Encoding rewriting in the ρ^{1st}-calculus

We are interested to build a ρ-term describing the reduction, in term rewriting, of term t w.r.t. a set of rewrite rules, but without knowing *a priori* the intermediate steps of the derivation of t. For this, we can use the ρ^{1st}_T-calculus and the operators defining *innermost* and *outermost* normalization strategies.

Proposition 3.1
Given a rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t{\downarrow} \in \mathcal{T}(\mathcal{F})$ such that t is normalized to $t{\downarrow}$ w.r.t. the set of rewrite rules $\mathcal{R}$. Then, $[im(\mathcal{R})](t)$ is ρ-reduced to a set containing the term $t{\downarrow}$.

PROOF. By induction on the number of reduction steps for the term t.

□

EXAMPLE 3.2
Let us consider a rewrite system $\mathcal{R}$ containing the rewrite rules $(x = x) \to True$ and $b \to a$. Then, the term $a = b$ reduces to $True$ in this rewrite system and a ρ-term reducing to $\{True\}$ can be built as shown in *Part I* or using the fixed-point operators.

In the former case the corresponding ρ-term is

$$[(x = x) \to True]([a = (b \to a)](a = b)).$$

For the latter approach we build the term

$$[im(\{(x = x) \to True, b \to a\})](a = b).$$

Since in this case we can obtain empty sets and additionally, sets with more than one element are obtained when equational matching is not unitary, a reduction strategy as presented in *Part I* should be used in order to ensure confluence. If no reduction strategy is used then undesired results can be obtained.

3.2 Encoding conditional rewriting

As shown before, any term rewriting reduction can be described by a reduction in the ρ-calculus. In this section we give a representation in the ρ-calculus of the conditional rewriting reductions. We will propose thus, methods for defining a ρ-term that contains all the information needed for reduction including the condition evaluation that is normally performed on the meta-level.

The main difficulty here resides in the fact that for conditional rewriting, the reduction relation is recursively applied in order to evaluate the condition when firing a conditional rule. We can use the same approach as our explicit description of non-conditional rewriting (see *Part I*) but the ρ-terms used in order to describe the conditional rewriting reduction become very complicated in this case. Instead, a detailed description by a concise ρ-term of the normalization process of the conditions can be obtained by using the normalization operators presented in the Section 2.5.

3.2.1 Definition of conditional rewriting

Many conditional rewriting relations have been designed and mainly differ in the way the conditions are understood [DO90]. We consider here the normal conditional rewriting defined as follows.

DEFINITION 3.3
A *normal* rewrite system $\mathcal{R}$ is composed of conditional rewrite rules of the form $(l \to r \text{ if } c)$ where l, r, c are elements of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ with variables satisfying the condition $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, and such that for each ground substitution σ satisfying $\mathcal{V}ar(c) \subseteq \mathcal{D}om(\sigma)$, the normal form under $\mathcal{R}$ of σc is either the boolean $True$ or $False$. Given a conditional rewrite system $\mathcal{R}$ composed of such rules, the application of the rewrite rule $(l \to r \text{ if } c)$ of $\mathcal{R}$ on a term t at occurrence m consists in:

(i) matching, using the substitution σ, the left-hand side of the rule against the term $t_{|m}$

(ii) normalizing the instantiated condition σc using $\mathcal{R}$ and, provided the resulting term is $True$,

(iii) replace $t_{|m}$ by σr in t.

This is denoted $t \longrightarrow_{[m]}^{l \to r \text{ if } c} t_{\lceil \sigma r \rceil_m}$.

3.2.2 Encoding

As we have mentioned, the main difficulty in the encoding of conditional rewriting is to make precise the evaluation process of the condition. In the case of normal rewriting, this means computing the normal form of the condition.

We denote by c_ρ the ρ-term that, when instantiated by the proper substitution (*i.e.* θc_ρ), normalizes to the term $\{u\}$ if the term c, instantiated accordingly (*i.e.* θc), is normalized into u in the rewrite theory $\mathcal{R}$. When the term c is a boolean condition and when the rewrite system is completely defined over the booleans [BR95], the term u should be one of the two constants $True$ or $False$.

If the reduction in a rewrite theory $\mathcal{R}$ is known, we can define, as in *Part I*, the ρ-term $c_\rho \triangleq [u_n](\ldots[u_1](c)\ldots)$ that evaluates to $\{u\}$, *i.e.* to $\{True\}$ or $\{False\}$. If c_ρ is the ρ-term describing the reduction of the term c then, the conditional rewrite rule $l \to r$ if c is represented by the ρ-term

$$l \to [\{True \to r, False \to \emptyset\}](c_\rho)$$

or even the simpler, but maybe less suggestive one,

$$l \to [True \to r](c_\rho).$$

In the case when c_ρ reduces to $\{False\}$, in the latter representation the matching fails and the result of the application is, as in the former one, the empty set. When c_ρ reduces to $\{True\}$, the result of the reduction is obviously the same in the two cases, *i.e.* the same as the application of $l \to r$.

By using the above representation, we can extend the Proposition given in *Part I* and show that any derivation in a conditional rewriting theory is representable by an appropriate ρ-term.

PROPOSITION 3.4
Given a conditional rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t' \in \mathcal{T}(\mathcal{F})$ such that $t \xrightarrow{*}_\mathcal{R} t'$. Then, there exist the ρ-terms $u_1, \ldots, u_n$ built using the rewrite rules in $\mathcal{R}$ and the intermediate steps in the derivation $t \xrightarrow{*}_\mathcal{R} t'$ such that we have $[u_n](\ldots[u_1](t)\ldots) \xrightarrow{*}_{\rho_\emptyset} \{t'\}$.

The construction approach used in *Part I* for unconditional rewriting is obviously not convenient and we need a method that allows us to build the ρ-term corresponding to a rewrite reduction without knowing *a priori* the reduction steps. In order to build the ρ-term c_ρ using only the term c and the rewrite rules of $\mathcal{R}$, we can use the normalization operators defined in Section 2. For example, we can define

$$c_\rho \triangleq [im(\mathcal{R})](c).$$

Example 3.5
Let us assume that the set of rules describing the order on integers is denoted by $\mathcal{R}_<$. We consider the rewrite rule $(f(x) \to g(x) \text{ if } x \geq 1)$ that applied to the term $f(2)$ reduces to $g(2)$ since x is instantiated by 2 and the condition $(2 \geq 1)$ reduces to $True$ by using the rewrite rule $(2 \geq 1) \to True$.

If we consider that the condition is normalized according to $\mathcal{R}_<$, then the corresponding reduction in the ρ-calculus is the following:

$$\begin{array}{ll} & [f(x) \to [True \to g(x)]([im(\mathcal{R}_<)](x \geq 1))](f(2)) \\ \longrightarrow_{Fire} & \{[True \to g(2)]([im(\mathcal{R}_<)](2 \geq 1))\} \\ \xrightarrow{*}_{\rho} & \{[True \to g(2)](\{True\})\} \\ \longrightarrow_{Batch} & \{\{[True \to g(2)](True)\}\} \\ \longrightarrow_{Fire} & \{\{\{g(2)\}\}\} \\ \xrightarrow{*}_{Flat} & \{g(2)\} \end{array}$$

The conditions of the rewrite rules can be normalized according to a set of conditional rewrite rules, including the current rule, and thus the definition of the ρ-rewrite rules representing this normalization is intrinsically recursive and cannot be realized only by using the operator im.

We use the fixed-point operator Θ described in Section 2.4 to represent the application of the same set of rewrite rules for the normalization of all the conditions.

Given a set of rewrite rules $\mathcal{R} = \mathcal{R}_n \cup \mathcal{R}_c$ where $\mathcal{R}_n$ and $\mathcal{R}_c$ represent the subset of non-conditional rewrite rules and respectively the subset of conditional rewrite rules of the form $(l \to r \text{ if } c)$. We define the term

$$R \triangleq f \to (y \to [im(\{l_i \to [True \to r_i]([f](c_i)) \mid i = 1 \dots m\} \cup \mathcal{R}_n)](y))$$

where $\mathcal{R}_c = \{l_i \to r_i \text{ if } c_i \mid i = 1 \dots m\}$, $\mathcal{R}_n = \{l'_i \to r'_i \mid i = 1 \dots n\}$ and respectively

$$IM(R) \triangleq [\Theta](R).$$

Thus, for describing the normalization of the term t w.r.t. the rewrite rules of $\mathcal{R}$ we use the ρ-term $[IM(R)](t)$.

The normalization strategy for the conditions is now abstracted by the variable f and since $IM(R) \triangleq [\Theta](R)$ is reduced to$[R]([\Theta](R))$ then this variable is instantiated at the beginning by $[\Theta](R)$ (*i.e.* $IM(R)$). Thus, not only the initial term but also the conditions are reduced according to $IM(R)$. This instantiation can be possibly reiterated if some conditional rules suppose the application of other conditional rules.

We obtain thus a result similar to Proposition 3.4 but with a method of construction for the corresponding ρ-term based only on the initial term and on the set of rewrite rules.

Proposition 3.6
Given a conditional rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t{\downarrow} \in \mathcal{T}(\mathcal{F})$ such that t is normalized to $t{\downarrow}$ w.r.t. the set of rewrite rules $\mathcal{R}$. Then, $[IM(\mathcal{R})](t)$ is ρ-reduced to a set containing the term $t{\downarrow}$.

Example 3.7
We consider the set of rewrite rules $\mathcal{R}$ containing the rewrite rule $(x = x) \to True$ and the conditional rewrite rules $(f(x) \to g(x) \text{ if } h(x) = b)$ and $(h(x) \to b \text{ if } x = a)$.

The term $f(a)$ reduces to $g(a)$ using the rewrite rules of $\mathcal{R}$ and we show below the corresponding reduction in ρ-calculus.

Using the method presented above we obtain the ρ-term:

$$\begin{array}{l} R \triangleq f \to (y \to [im(\{f(x) \to [True \to g(x)]([f](h(x) = b)), \\ \qquad\qquad\qquad\qquad h(x) \to [True \to b]([f](x = a)), \\ \qquad\qquad\qquad\qquad (x = x) \to True \\ \qquad\qquad\qquad \})](y)) \end{array}$$

We show the main steps in the reduction of the term $[IM(R)](f(a))$. We obtain immediately the reduction

$$[IM(R)](f(a)) \triangleq [[\Theta](R)](f(a)) \quad \xrightarrow{*}_{\rho} \quad [[R]([\Theta](R))](f(a)) \triangleq [[R](IM(R))](f(a))$$

and the final result is the same as the one obtained for the term

$$\begin{array}{l} [im(\{f(x) \to [True \to g(x)]([IM(R)](h(x) = b)), \\ \qquad h(x) \to [True \to b]([IM(R)](x = a)), \\ \qquad (x = x) \to True \\ \})](f(a))) \end{array}$$

and thus for

$[f(x) \to [True \to g(x)]([IM(R)](h(x) = b))](f(a))$

$\xrightarrow{*}_{\rho} \quad \{[True \to g(a)]([IM(R)](h(a) = b))\}$

For the term $[IM(R)](h(a) = b)$ we proceed as previously and thus, we have to reduce the term

$$\begin{array}{l} [im(\{f(x) \to [True \to g(x)]([IM(R)](h(x) = b)), \\ \qquad h(x) \to [True \to b]([IM(R)](x = a)), \\ \qquad (x = x) \to True\} \\ )](h(a) = b) \end{array}$$

with the intermediate reduction

$$[h(x) \to [True \to b]([IM(R)](x = a))](h(a)) \quad \xrightarrow{*}_{\rho} \quad \{[True \to b]([IM(R)](a = a))\}$$

Since we easily obtain $[IM(R)](a = a) \xrightarrow{*}_{\rho} \{True\}$ then, the previous term is reduced to $\{[True \to b](\{True\})\} \xrightarrow{*}_{\rho} \{b\}$ and we have

$$[IM(R)](h(a) = b) \quad \xrightarrow{*}_{\rho} \quad [im(\ldots)](\{b\} = b) \quad \xrightarrow{*}_{\rho} \quad \{True\}$$

We come back to the reduction of the initial term and we get

$$\{[True \to g(a)]([IM(R)](h(a) = b))\} \quad \xrightarrow{*}_{\rho} \quad \{[True \to g(a)](\{True\})\} \quad \xrightarrow{*}_{\rho} \quad \{g(a)\}$$

We have thus obtained the same result as in conditional term rewriting.

Starting from the results presented in this section we will give in the next section a representation of the more elaborated rewrite rules used in ELAN, a language based on conditional rewrite rules with local assignments.

4 The rewriting calculus as a semantics of ELAN

4.1 ELAN*'s rewrite rules*

ELAN (a name that expresses the dynamism of the arrow), is an environment for specifying and prototyping deduction systems in a language based on labeled conditional rewrite rules and strategies to control rule application. The ELAN system offers a compiler and an interpreter of the language. The ELAN language allows us to describe in a natural and elegant way various deduction systems [Vit94, KKV95, BKK+96]. It has been experimented on several non-trivial applications ranging from decision procedures, constraint solvers [Cas98], logic programming [KR98] and automated theorem proving [CK97] but also specification and exhaustive verification of authentication protocols [Cir99].

ELAN's rewrite rules are conditional rewrite rules with local assignments. The local assignments are let-like constructions that allow applications of strategies to some terms. The general syntax of an ELAN rule is:

$$[\ell] \quad l \Rightarrow r \qquad [\ \textbf{if}\ \ cond \quad | \quad \textbf{where}\ \ y := (S)u\ ]^* \quad end$$

where *cond* is an ELAN expression that can be reduced to a boolean value. If all the conditions are reduced to the `true` value and all local variables (*e.g.* y) are assigned with success (*i.e.* the application of the strategy from the right-hand side of the local assignment does not fail) then the rewrite rule can be applied.

We should notice that the square brackets ([ ]) in ELAN are used to indicate the label of the rule and should be distinguished from the square brackets of the ρ-calculus that represent the application of a rewrite rule (ρ-term).

A partial semantics could be given to an ELAN program using rewriting logic [Mes92, BKKM99], but more conveniently all ELAN's rules (and not only the conditional ones) and strategies can be expressed using the ρ-calculus and thus an ELAN program is just a a ρ-term. The results of the evaluation of this ρ-term correspond to all the possible results of the execution of the initial ELAN program.

Example 4.1
An example of a labeled ELAN rule describing a possible naive way to search the minimal element of a list by sorting the list and taking the first element is the following:

```
[min-rule]  min(l)   =>   m
                 if l !=  nil
                 where sl := (sort) l
                 where m  := () head(sl)  end
```

The strategy `sort` can be any sorting strategy. The operator `head` is supposed to be described by a confluent and terminating set of unlabeled rewrite rules. Thus, `sl` is assigned the result of the application of a given set of labeled rules guided by the strategy `(sort)`, while `m` is assigned the result of the application of a given set of unlabeled rules guided by the strategy `()` (*i.e.* the implicit built-in *innermost* strategy).

The evaluation strategy used for evaluating the conditions is a leftmost innermost standard rewriting strategy.

The non-determinism is handled mainly by two basic strategy operators: `dont care choose` (denoted `dc`$(s_1, \ldots, s_n)$) that returns the results of at most one non-deterministicly chosen unfailing strategy from its arguments and `dont know choose` (denoted `dk`$(s_1, \ldots, s_n)$) that returns all the possible results. A variant of the `dont care choose` operator is the `first choose` operator (denoted `first`$(s_1, \ldots, s_n)$) that returns the results of the first unfailing strategy from its arguments.

Several strategy operators implemented in ELAN allow us a simple and concise description of user defined strategies. For example, the concatenation operator denoted "`;`" builds the sequential composition of two strategies s_1 and s_2. The strategy $s_1; s_2$ fails if s_1 fails, otherwise it returns all results (maybe none) of s_2 applied to the results of s_1. Using the operator `repeat*` we can describe the repeated application of a given strategy. Thus, `repeat*`(s) iterates the strategy s until it fails and then returns the last obtained result.

Any rule in ELAN is considered as a basic strategy and several other strategy operators are available for describing the computations. Here is a simple example illustrating the way the `first` and `dk` strategies work.

Example 4.2
If the strategy `dk(x=>x+1,x=>x+2)` is applied to the term a, ELAN provides two results: $a+1$ and $a+2$. When the strategy `first(x=>x+1,x=>x+2)` is applied to the same term only the $a+1$ result is obtained. The strategy `first(b=>b+1,a=>a+2)` applied to the term a yields the result $a+2$.

Using non-deterministic strategies, we can explore exhaustively the search space of a given problem and find paths described by some specific properties.

For example, for proving the correctness of the Needham-Schroeder authentication protocol [NS78] we look for possible attacks among all the behaviors during a session. In Example 4.3 we present just one of the rules of the protocol and we give the strategy looking for all the possible attacks, a more detailed description of the implementation is given in [Cir99].

Example 4.3
We consider the rewrite rules describing the Needham-Schroeder authentication protocol that aims to establish a mutual authentication between an initiator and a responder that communicate via an insecure network (*i.e.* in presence of intruders).

The strategy looking for possible attacks applies repeatedly and non-deterministicly all the rewrite rules describing the behavior of the protocol (*e.g.* initiate) and of the intruder (*e.g.* intruder) and selects only those results representing an attack.

```
[]attStrat => repeat*(
                        dk( initiate, ..., intruder)
              );
              attackFound      end
```

The non-deterministic application is described with the operator `dk`. The result of the strategy `repeat*(...)` is the set of all possible behaviors in a protocol session where messages can be intercepted or faked by an intruder. The strategy `attackFound` just checks if the term received as input represents an attack (by trying to apply the rewrite rules corresponding to the negation of the desired invariants) and therefore selects from the previous set of results only those representing an attack.

4.2 The ρ-calculus representation of ELAN *rules*

The rules of the system ELAN can be expressed using the ρ-calculus. A rule with no conditions and no local assignments $l \Rightarrow r$ is represented by $l \to r$ and a conditional rule is expressed as in Section 3.2.

4.2.1 Rules with local assignments

The ELAN rewrite rules with local assignments but without conditions of the form

$$[\ell] \quad l(x) \Rightarrow \quad r(x,y) \quad \texttt{where} \;\; y \;:= (S)u$$

can be represented by the ρ-term

$$l(x) \to r(x, [S_\rho](u))$$

or the ρ-term

$$l(x) \to [y \to r(x,y)]([S_\rho](u))$$

with S_ρ, the ρ-term corresponding to the strategy S in the ρ-calculus.

The first representation syntactically replaces all variables of the right-hand side of the rewrite rule defined in a local assignment with the term which instantiates the respective variable. In the second representation, each variable defined in a local assignment is bound in a ρ-rewrite rule which is applied to the corresponding term.

Example 4.4
The ELAN rule

```
[deriveSum] p_1 + p_2  =>  p_1' + p_2'
                           where p_1' := (derive)p_1
                           where p_2' := (derive)p_2       end
```

can be represented by one of the following two ρ-terms
$p_1 + p_2 \to [derive](p_1) + [derive](p_2)$,
$p_1 + p_2 \to [p_1' \to [p_2' \to p_1' + p_2'][derive](p_2))][derive](p_1))$.

At this moment one can notice the usefulness of free variables in the rewrite rules. The latter representation of an ELAN rule with local assignments would not be possible if the variable p_1' was not allowed to be free in the ρ-rule $p_2' \to p_1' + p_2'$. The free variables in the right-hand side of a ρ-rewrite-rule also enables the parameterization of rewrite rules by strategies as in $y \to [f(x) \to [y](x)](f(a))$ where the strategy to be applied on x is not known in the rule $f(x) \to [y](x)$.

Example 4.5
We consider the ELAN rule

```
[deriveSum] x  =>  y + y
                   where y := (derive)x       end
```

Let us consider that the strategy `derive` is `dk(a=>b,a=>c)`. Then, the application of the strategy `derive` to the term a gives the two results b and c. Thus, the application of the rule `deriveSum` to the term a provides non-deterministically one of the four results $b + b$, $b + c$, $c + b$, $c + c$.

The ρ-representation of this rule is

$$x \to [\{a \to b, a \to c\}](x) + [\{a \to b, a \to c\}](x)$$

that applied to a reduces as follows

$$\begin{array}{ll}
 & [x \to [\{a \to b, a \to c\}](x) + [\{a \to b, a \to c\}](x)](a) \\
\longrightarrow_{Fire} & \{[\{a \to b, a \to c\}](a) + [\{a \to b, a \to c\}](a)\} \\
\overset{*}{\longrightarrow}_{Distrib} & \{\{[a \to b](a), [a \to c](a)\} + \{[a \to b](a), [a \to c](a)\}\} \\
\overset{*}{\longrightarrow}_{Fire} & \{\{\{b\}, \{c\}\} + \{\{b\}, \{c\}\}\} \\
\longrightarrow_{Flat} & \{\{b, c\} + \{b, c\}\} \\
\longrightarrow_{OpOnSet} & \{\{b + \{b, c\}, c + \{b, c\}\}\} \\
\longrightarrow_{OpOnSet} & \{\{\{b + b, b + c\}, \{c + b, c + c\}\}\} \\
\longrightarrow_{OpOnSet} & \{\{\{b + b, b + c\}, \{c + b, c + c\}\}\} \\
\overset{*}{\longrightarrow}_{Flat} & \{b + b, b + c, c + b, c + c\}
\end{array}$$

This set represents exactly the four results obtained in ELAN.

If we consider more general ELAN rules containing local assignments as well as conditions on the local variables, the combination of the methods used for conditional rules and rules with local assignments should be done carefully. If we had used a representation closed to the first one from Example 4.4 we would have obtained some incorrect results as in Example 4.6.

Example 4.6
We consider the description of an automaton by a set of rewrite rules, each one describing the transition from a state to another. The potential execution of a double transition from an initial state in a final state passing by a non-final intermediate state, can be described by the following ELAN rule:

```
[double] x => next(y)
                    where y := (dk(s1 => s2,s1 => s3)) x
                    if nf(y)
end
```

The term `next(y)` represents the state obtained by carrying out a transition from `y` and this behavior can be easily represented in ELAN by a set of unlabeled rules describing the operator `nf`. We note by $\mathcal{R}_f$ the set of rewrite rules describing the final states and we suppose that `s2` is a final state but `s3` is not.

By using the first representation approach of a rule with local assignments and the coding method for conditional rules presented in Section 3.2, we obtain the ρ-term corresponding to the previous ELAN rule:

$$x \to [True \to next([\{s1 \to s2, s1 \to s3\}](x))]([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](x))))$$

This term applied to $s1$ leads to the following reduction

$$\begin{array}{l}
[x \to [True \to next([\{s1 \to s2, s1 \to s3\}](x))]([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](x))))](s1) \\
\longrightarrow_{\rho} \{[True \to next([\{s1 \to s2, s1 \to s3\}](s1))]([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](s1))))\} \\
\overset{*}{\longrightarrow}_{\rho} \{[True \to next(\{s2, s3\})]([im(\mathcal{R}_f)](nf(\{s2, s3\})))\}
\end{array}$$

$\xrightarrow{*}_\rho \{[True \to \{next(s2), next(s3)\}]([im(\mathcal{R}_f)](\{nf(s2), nf(s3)\}))\}$
$\xrightarrow{*}_\rho \{[True \to \{next(s2), next(s3)\}](\{False, True\})\}$
$\xrightarrow{*}_\rho \{\{[True \to \{next(s2), next(s3)\}](False), [True \to \{next(s2), next(s3)\}](True)\}\}$
$\xrightarrow{*}_\rho \{\emptyset, [True \to \{next(s2), next(s3)\}](True)\}$
$\xrightarrow{*}_\rho \{\emptyset, \{next(s2), next(s3)\}\}$
$\xrightarrow{*}_\rho \{next(s2), next(s3)\}$

while in ELAN we obtain the only result `next(s3)` that would be represented by the ρ-term $\{next(s3)\}$.

The problem in the Example 4.6 is the double evaluation of the term $[\{s1 \to s2, s1 \to s3\}](s1)$ replacing the local variable y: once in the condition and once in the right-hand side of the rule. If this term is evaluated to a set with more than one element and one of its elements satisfies the condition, then this set replaces the corresponding variables in the right-hand side of the rule, while only the subset of elements satisfying the condition should be considered. Therefore, we need a mechanism that evaluates only once each of the local assignments of a rule.

We use an approach combining the second representation approach of a rule with local assignments and the ρ-representation of conditional rules. Without losing generality, we consider that an ELAN rule that has the following form:

$[label] \quad l \Longrightarrow r_{\lceil x \rceil_q}$
`where` $x := (s)t$
`if` $C_{\lceil x \rceil_p}$
`end`

Then, the ELAN rule presented above is expressed as the ρ-term

$$l \to [x \to [\{True \to r_{\lceil x \rceil_q}, False \to \emptyset\}]([im(\mathcal{R})](C_{\lceil x \rceil_p}))]([s](t))$$

or the simpler one

$$l \to [x \to [True \to r_{\lceil x \rceil_q}]([im(\mathcal{R})](C_{\lceil x \rceil_p}))]([s](t))$$

where $\mathcal{R}$ represents the set of rewrite rules modulo which we normalize the conditions.

In order to simplify the presentation we supposed that the rules of the set $\mathcal{R}$ are rewrite rules of the form $l \to r$ and thus, the operator im is sufficient to define normalization w.r.t. such a set. If we consider conditional unlabeled rules, then the operator IM must be employed.

The way the transformation is applied to an ELAN rewrite rule and the corresponding reduction are illustrated by taking again the Example 4.6 and considering the new representation.

Example 4.7
The ELAN rewrite rule from Example 4.6 is represented by the ρ-term

$$x \to [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](x))$$

that, applied to the term $s1$ leads to the following reduction

$$
\begin{array}{ll}
& [x \to [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](x))](s1) \\
\longrightarrow_{Fire} & \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](s1))\} \\
\xrightarrow{*}_{\rho} & \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](\{s2, s3\})\} \\
\xrightarrow{*}_{\rho} & \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](s2), \\
& \quad [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](s3)\} \\
\xrightarrow{*}_{Fire} & \{\{[True \to next(s2)]([im(\mathcal{R}_f)](nf(s2)))\}, \\
& \quad \{[True \to next(s3)]([im(\mathcal{R}_f)](nf(s3)))\}\} \\
\xrightarrow{*}_{\rho} & \{[True \to next(s2)](False), [True \to next(s3)](True)\} \\
\xrightarrow{*}_{\rho} & \{\emptyset, \{next(s3)\}\} \\
\xrightarrow{*}_{\rho} & \{next(s3)\}
\end{array}
$$

that is the representation of the result obtained in ELAN.

The same result as in Example 4.6 is obtained if the evaluation rule $Fire$ is applied before the distribution of the set $\{s2, s3\}$. But the confluent strategies presented in *Part I* forbid such a reduction and thus, the correct result is obtained.

This latter representation not only allows a correct transformation of ELAN reductions in ρ-reductions but gives also a hint on the implementation details of such rewrite rules. On one hand the implementation should ensure the correctness of the result and on the other hand it should take into account the efficiency problems. For instance, the representation used in Example 4.5 is correct but obviously less efficient than a representation as in Example 4.7 and this is due to the double evaluation of the same application.

The ELAN evaluation mechanism is more complex than presented above since it distinguishes between labeled rewrite rules and unlabeled rewrite rules. The unlabeled rewrite rules are used to normalize the result of all the applications of a labeled rewrite rule to a term. When evaluating a local assignment `where v:=(S) t` of an ELAN rewrite rule, the term `t` is first normalized according to the specified set of unlabeled rewrite rules and then the strategy `S` is applied to its normal form. Moreover, each time a labeled rewrite rule is applied to a term, the ELAN evaluation mechanism normalizes the result of its application with respect to the set of unlabeled rewrite rules.

Hence, the ELAN rewrite rule from Example 4.6 should be represented in the ρ-calculus by the term

$$
\begin{array}{l}
x \to [im(\mathcal{R}_f)]([y \to [True \to next(y)] \\
\qquad\qquad ([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}]([im(\mathcal{R}_f)](x))))
\end{array}
$$

where $\mathcal{R}_f$ represents the set of (unlabeled) rewrite rules modulo which we normalize the local assignments.

4.2.2 General strategies in the local assignments

Until now we have considered in the local assignments of a rule only strategies that do not use the respective rewrite rule. The representation of an ELAN rule with local calls to strategies defined by using this rule must be parameterized by the definition of the respective strategies. For example, a rule with local assignments of the form

$$[label] \qquad l \Longrightarrow r \ \texttt{where}\ x := (s)t$$

is represented by the ρ-term

$$label(f) \triangleq l \to [x \to r]([[f](s)](t))$$

where the free variable f will be instantiated by the set of strategies of the program containing the rule labeled by *label*.

4.2.3 ELAN strategies and programs

The elementary ELAN strategies has, in most of the cases, a direct representation in the ρ-calculus. The identity (`id`) and the failure (`fail`) as well as the concatenation (`;`) are directly represented in the ρ-calculus by the ρ-operators id, $fail$ and ";" respectively, defined in Section 2.1. The strategy $\texttt{dk}(S_1, \ldots, S_n)$ is represented in the ρ-calculus by the set $\{S_1, \ldots, S_n\}$ and the strategy $\texttt{first}(S_1, \ldots, S_n)$ by the ρ-term $first(S_1, \ldots, S_n)$ defined in Section 2.2. The iteration strategy operator `repeat*` is easily represented by using the ρ-operator $repeat*$.

Strategies can be used in the evaluation of the local assignments and these strategies are expressed using rewrite rules. Therefore, the ELAN strategies can be represented by ρ-terms in the same way as the ELAN rewrite rules.

Example 4.8
The ELAN strategy `attStrat` used in Example 4.3 is immediately represented by the ρ-term

$$attStrat_\rho \to repeat*(\{initiate_\rho, \ldots, intruder_\rho\}); attackFound_\rho$$

where we suppose that $initiate_\rho$, $intruder_\rho$, $attackFound_\rho$ are the representations in ρ-calculus of the corresponding ELAN strategies.

For the representation of the user-defined strategies in an ELAN program we use an approach based on the fixed-point operator and similar to that used in the case of conditional rules in Section 3.2. If we consider an ELAN program containing the strategies $S_1, \ldots, S_n$ and a set of labeled rules, then the ρ-term representing the program is

$$P \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (y \to [\{S_i \to Body_i \mid i = 1 \ldots n\}](y))$$

and $Body_i$ represent the right-hand sides of the strategies with each strategy S_i replaced by $[f](S_i)$, each rule label replaced by the ρ-representation of the rule and each ELAN strategy operator replaced by its correspondent in the ρ-calculus.

To sum-up, we present the transformation of an ELAN program in a ρ-term.

Definition 4.9
We consider an ELAN without importations.

1. The signature of the corresponding ρ-calculus is obtained from the operator declarations of the ELAN program.
2. Starting from unlabeled rules of the form

$[]$ $\quad l_i(\overline{x}) \Longrightarrow r_i(\overline{x},\overline{y})$
$\quad$ `where` $(sort)\ u_i(\overline{y}) := ()t_i(\overline{x})$
$\quad$ `if` $c_i(\overline{x},\overline{y})$
`end`

we build the term

$$R_{nn} \triangleq\ f \to (z \to [im(\ \{l_i(\overline{x}) \to [u_i(\overline{y}) \to [True \to r_i(\overline{x},\overline{y})]([f](c_i(\overline{x},\overline{y})))](t_i(\overline{x}))\ |\ i = 1 \ldots n\})](z))$$

The *innermost* normalization w.r.t. the set of unlabeled rules is represented by the term

$$IM_{nn} \triangleq [\Theta](R_{nn})$$

The encoding is extended in an incremental way to rules containing several conditions and local assignments. The encoding can be simplified if the program does not contain unlabeled conditional rules; in this case the term IM_{nn} becomes

$$IM_{nn} \triangleq im(\{l_i(\overline{x}) \to [u_i(\overline{y}) \to r_i(\overline{x},\overline{y})](t_i(\overline{x}))\ |\ i = 1 \ldots n\})$$

where the rules with local assignments can be simplified to elementary rules.

3. For each labeled rule of the form

$[label]$ $\quad l(\overline{x}) \Longrightarrow r(\overline{x},\overline{y})$
$\quad$ `where` $(sort)\ u(\overline{y}) := (s)t(\overline{x})$
$\quad$ `if` $c(\overline{x},\overline{y})$
`end`

we build the term

$$label(f) \triangleq\ f \to (l(\overline{x}) \to [IM_{nn}](\ [u(\overline{y}) \to [True \to r(\overline{x},\overline{y})]([IM_{nn}](c(\overline{x},\overline{y})))](\,[[f](s)]([IM_{nn}](t(\overline{x})))))$$

4. For each strategy of the form

$[]$ $\quad S \Longrightarrow Body$
`end`

we build the term

$$S \to BodyRho(f)$$

where $BodyRho$ represents the right-hand side $Body$ of the strategy with each strategy symbol S_i replaced by $[f](S_i)$, each rule label $label$ replaced by the ρ-representation $label(f)$ of the rule and each ELAN strategy operator replaced by its correspondent in the ρ-calculus.

The ELAN program defining the strategies $S_1, \ldots, S_n$ is represented by the ρ-term

$$P \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (z \to [\{S_i \to BodyRho_i(f) \mid i = 1 \ldots n\}](z))$$

and $BodyRho_i(f)$ represents the encoding of the strategy S_i.

The application of a strategy $\mathcal{S}$ of an ELAN program $\mathcal{P}$ to a term t is represented by the ρ-term $[[P](s)](t)$ where P is the ρ-term representing the program $\mathcal{P}$ and s is the name of the strategy $\mathcal{S}$. If the execution of the program $\mathcal{P}$ for evaluating the term t according to the strategy $\mathcal{S}$ leads to the results $u_1, \ldots, u_n$, then the ρ-term $[[P](s)](t)$ is reduced to the set term $\{u_1, \ldots, u_n\}$.

In Example 4.10 we present an ELAN module and the ρ-interpretations of all the rules and strategies and thus, of the ELAN program.

Example 4.10
The module `automaton` describes an automaton with the states `s1,s2,s3,s4,s5` and with the non-deterministic transitions described by a set of rules containing the rules labeled with `r12,r13,r25,r32,r34,r41`. The operator `next` defines the next state in a deterministic manner and its behavior is described by a set of unlabeled rules. The states can be "final" (`final`) or "closed" (`closed`). The double transitions with an intermediate non-final and non-closed state are described by the rules `double_f` and respectively `double_c`.

```
module  automaton
import global bool;end
sort state ;end
operators global
    s1,s2,s3,s4,s5 : state;
    next(@) : (state)  state;
    final(@) : (state)  bool;
    closed(@) : (state)  bool;
end
stratop  global
    follow   : <state -> state>   bs;
    gen_double   : <state -> state>   bs;
    cond_double   : <state -> state>   bs;
end
rules for bool
global
 [] final(s_1) => false   end         [] closed(s_1) => false   end
 [] final(s_2) => true    end         [] closed(s_2) => false   end
 [] final(s_3) => false   end         [] closed(s_3) => true    end
 [] final(s_4) => false   end         [] closed(s_4) => true    end
 [] final(s_5) => true    end         [] closed(s_5) => true    end
end
rules for state
    x,y : state;
```

```
global
  [r12] s1 => s2   end                  [] next(s1) => s3   end
  [r13] s1 => s3   end                  [] next(s2) => s5   end
  [r25] s2 => s5   end                  [] next(s3) => s2   end
  [r32] s3 => s2   end                  [] next(s4) => s1   end
  [r34] s3 => s4   end                  [] next(s5) => s5   end
  [r41] s4 => s1   end

  [double_f]  x => next(y)
                       where y := (follow) x
                       if not final(y)              end
  [double_c]  x => next(y)
                       where y := (follow) x
                       if not closed(y)             end
end

strategies for state
implicit
    []follow         =>   dk(r12,r13,r25,r32,r34,r41)          end
    []gen_double     =>   follow;follow                        end
    []cond_double    =>   dk(double_f,double_c)                end
end
end
```

We denote by B the set of unlabeled rules defined in the imported modules `bool` and describing operations on booleans.

The set of unlabeled rules from the module `automaton` are represented by the ρ-term

$$\begin{array}{rcl} R & \triangleq & \{next(s1) \to s3, \ldots, next(s5) \to s5, \\ & & final(s1) \to false, \ldots, final(s5) \to true, \\ & & closed(s1) \to false, \ldots, closed(s5) \to true\} \end{array}$$

and we note $RC = R \cup B$.

The rules labeled with `double_f` and `double_c` are represented by the ρ-rules

$$\begin{array}{rcl} double\_f(f) & \triangleq & x \to [im(RC)](\ [y \to [True \to next(y)]([im(RC)](not\ final(y)))] \\ & & \qquad ([[f](follow)]([im(RC)](x)))) \end{array}$$

and respectively

$$\begin{array}{rcl} double\_c(f) & \triangleq & x \to [im(RC)](\ [y \to [True \to next(y)]([im(RC)](not\ closed(y)))] \\ & & \qquad ([[f](follow)]([im(RC)](x)))) \end{array}$$

The strategies from the module `automaton` are represented by the ρ-terms

$$\begin{array}{lcl} follow & \triangleq & follow \to \{s1 \to s2, s1 \to s3, s2 \to s5, s3 \to s2, s3 \to s4, s4 \to s1\} \\ gen\_double(f) & \triangleq & gen\_double \to [f](follow); [f](follow) \\ cond\_double(f) & \triangleq & cond\_double \to \{double\_f(f), double\_c(f)\} \end{array}$$

and we obtain the term representing the ELAN program `automaton`

$$automaton \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (y \to [\{follow, gen\_double(f), cond\_double(f)\}](y))$$

The execution of the program `automaton` for evaluating the term `s1` with the strategy `cond_double` corresponds to the reduction of the term

$$[[automaton](cond\_double)](s1)$$

In ELAN, we obtain for such an execution the results `2` and `5` and the reduction of the corresponding ρ-term leads to the set $\{2, 5\}$.

In Example 4.10 we presented a relatively simple ELAN module but, representative for the main features of the ELAN language. Following the same methodology, more complicated rules and strategies can be handled.

Notice that this provides, in particular, a very precise description of all the rewriting primitives, including the semantics of the conditional rewriting used by the language. To the best of our knowledge, this is the first *explicit* and *full* description of a rewrite based programming language.

5 Conclusion

Using the ρ^{1st}-calculus, an extension of the ρ-calculus, appropriate definitions of term traversal operators and of a fixed-point operator can be given. This enables us to apply repeatedly a (set of) rewrite rule(s) and consequently to define a ρ-term representing the normalization according to a set of rewrite rules. Starting from this representation we showed how the ρ^{1st}-calculus can be used to define conditional rewriting and to give a semantics to ELAN modules. Of course, this could be applied to many other frameworks, including rewrite based languages like ASF+SDF, ML, Maude, Stratego or CafeOBJ but also production systems and non-deterministic transition systems.

Starting from these first results on the rewriting calculus, we have already explored, in subsequent papers, two different directions: the ρ-calculus with explicit substitutions and typed rewriting calculi. In [Cir00] we have proposed a version of the calculus where the substitution application is described at the same level as the other evaluation rules. Starting from the λ-calculus with explicit substitutions, and in particular the $\lambda\sigma_{\Uparrow}$-calculus, we developed the ρ-calculus with explicit substitutions, called the $\rho\sigma$-calculus and we showed that the $\rho\sigma$-calculus is confluent under the same conditions as the $\rho_{\emptyset}$-calculus. Indeed, what makes the explicit substitution setting even more interesting than in the case of λ-calculus is that not only the substitution and therefore renaming mechanism is handled explicitly, but the substitution itself is explicitly represented. This is extremely useful since computing a substitution could be very expensive like for associativity-commutativity where the matching algorithm is exponential in the size of terms. Moreover, since a derivation may fail (like when searching for the right instance of a conditional), memorizing the substitution is mandatory. This allows us in particular to use the ρ-calculus with explicit substitutions as the language to describe proof terms of ELAN computations.

The ρ-calculus is not terminating in the untyped case. In order to recover this property we have imposed in [CK00] a more strict discipline on the ρ-term formation by introducing a type for each term. We presented a type system for the $\rho_\emptyset$-calculus and we showed that it has the subject reduction and strong normalization properties, *i.e.* that the reduction of any well-typed term is terminating and preserves the type of the initial term. Additionally, we have given a new presentation *à la Church* to the ρ-calculus [CKL01b], together with nine (8+1) type systems which can be placed in a *ρ-cube* that extends the λ-cube of Barendregt. Quite interestingly, this typed calculus uses only one abstractor, namely the rule arrow. It provides therefore a solution to the identification of the λ and Π abstractors.

We used the sets to represent the non-determinism and we mentioned that other structures can be used. For example, if we want to represent all the results of an application and not only the different results, then multisets must be used and if the order of the results is significant, then a list structure is more suitable. We have thus started the study of another description of the ρ-calculus having as parameter not only the matching theory but also the structure used for the results and we have already shown its expressive power [CKL01a]. More precisely, we analyzed the correspondence between the ρ-calculus and two object oriented calculi: the *"Object Calculus"* of Abadi and Cardelli [AC96] and the *"Lambda Calculus of Objects"* of Fisher, Honsell and Mitchell [FHM94]. The approach that we proposed allows the representation of objects in the style of the two mentioned calculi but also of more elaborate objects whose behavior is described by using the matching power.

As a new emergent framework, the ρ_T-calculus offers an original view point on rewriting and higher-order logic and it opens new challenges to further understand related topics. First, to go further in the study and the use of the ρ_T-calculus for the combination of first-order and higher-order paradigms, the investigation of the relationship of this calculus with higher-order rewrite concepts like CRS and HOR [vOvR93] should be deepened. Second, several directions should be investigated, amongst them, we can mention the following:

- The analysis of the properties of the ρ_T-calculus with a matching theory T more elaborate than syntactic matching.
- A generic description of the conditions which must be imposed for the matching theory T in order to obtain the confluence and the termination of the ρ_T-calculus should be defined and then, show that these conditions are satisfied for particular theories such as associativity and commutativity.
- The models of the rewriting calculus should be defined, studied and compared with the ones of the algebraic as well as higher-order structures.
- As mentioned previously, we conjecture that the ρ^{1st}-calculus can not be expressed in the ρ-calculus because of the semantics of the empty set as rule application failure.

Finally, from the practical point of view, the various instances of the ρ-calculus must be further implemented and used as rewriting tools. We have already realized an implementation in ELAN of the $\rho_\emptyset$-calculus and we experimented with various evaluation strategies. This implementation could be further used in order to define

object oriented paradigms. Dually, an object oriented version of the ELAN language has been realized [DK00], with a semantics given by the rewriting calculus.

This shows that this new calculus is very attractive in terms of semantics as well as unifying capabilities and we believe that it can serve as a basic tool for the integration of semantic and logical frameworks.

Acknowledgments

We would like to thank Hélène Kirchner, Pierre-Etienne Moreau and Christophe Ringeissen from the Protheo Team for the useful interactions we had on the topics of this paper, Vincent van Oostrom for suggestions and pointers to the literature, Roberto Bruni and David Wolfram for their detailed and very useful comments on a preliminary version of this work and Delia Kesner for fruitful discussions. We are grateful to Luigi Liquori for many comments and exciting discussions on the ρ-calculus and its applications. Many thanks also to Thérèse Hardin and Nachum Dershowitz for their interest, encouragements and helpful suggestions for improvement. Finally special thanks are due to the referees for the very complete and careful reading of the paper as well as constructive and useful remarks.

References

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer Verlag, 1996.

[Bar84] H. P. 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.

[BKK+96] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[BKKM99] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. *ELAN* from the rewriting logic point of view. Research report, LORIA, November 1999.

[BR95] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

[Cas98] C. Castro. *Une approche déductive de la résolution de problèmes de satisfaction de contraintes*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, France, 1998.

[CELM96] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[Cir99] H. Cirstea. Specifying Authentication Protocols Using ELAN. In *Workshop on Modelling and Verification*, Besancon, France, December 1999.

[Cir00] H. Cirstea. *Calcul de réécriture : fondements et applications*. Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[CK97] H. Cirstea and C. Kirchner. Theorem Proving Using Computational Systems: The Case of the B Predicate Prover. In *Workshop CCL'97*, Schloβ Dagstuhl, Germany, September 1997.

[CK00] H. Cirstea and C. Kirchner. The Simply Typed Rewriting Calculus. In *3rd International Workshop on Rewriting Logic and its Applications*, Kanazawa, Japan, September 2000. Electronic Notes in Theoretical Computer Science.

[CKL01a] 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.

[CKL01b] H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. In F. Honsell, editor, *Foundations*

of Software Science and Computation Structures, Lecture Notes in Computer Science, Genova, Italy, April 2001.

[DK00] H. Dubois and H. Kirchner. Objects, rules and strategies in `ELAN`. In *Proceedings of the second AMAST workshop on Algebraic Methods in Language Processing, Iowa City, Iowa, USA*, May 2000.

[DO90] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[FHM94] K. Fisher, F. Honsell, and J. C. Mitchell. A Lambda Calculus of Objects and Method Specializatio n. *Nordic Journal of Computing*, 1(1):3–37, 1994.

[FN97] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In *Proceedings of the 1st IEEE Int. Conference on Formal Engineering Methods*, 1997.

[GMW79] M. Gordon, A. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, New York (NY, USA), 1979.

[KKV95] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.

[Kli93] P. Klint. The ASF+SDF Meta-environment User's Guide. Technical report, CWI, 1993.

[KR98] C. Kirchner and C. Ringeissen. Rule-Based Constraint Programming. *Fundamenta Informaticae*, 34(3):225–262, September 1998.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[Mil84] R. Milner. A proposal for standard ML. In *Proceedings ACM Conference on LISP and Functional Programming*, 1984.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pro01] Protheo Team. The ELAN home page. WWW Page, 2001. `http://elan.loria.fr`.

[Tur37] A. M. Turing. The $\wp$-functions in λ-K-conversion. *The Journal of Symbolic Logic*, 2:164, 1937.

[VeAB98] E. Visser and Z. el Abidine Benaissa. A core language for rewriting. In C. Kirchner and H. Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume16.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science.

[Vis99] E. Visser. Strategic pattern matching. In P. Narendran and M. Rusinowitch, editors, *Rewriting Techniques and Applications (RTA'99)*, volume 1631 of *Lecture Notes in Computer Science*, pages 30–44, Trento, Italy, July 1999. Springer-Verlag.

[Vit94] M. Vittek. ELAN*: Un cadre logique pour le prototypage de langages de programmation avec contraintes*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, October 1994.

[vOvR93] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In *HOA'93*, volume 816 of *Lecture Notes in Computer Science*, pages 276–304. Springer-Verlag, 1993.

Contents

Received