Combining Higher-Order and First-Order Computation Using ρ-calculus: Towards a semantics of ELAN

Horatiu Cirstea Claude Kirchner

LORIA and INRIA
615, rue du Jardin Botanique, B.P. 101
54602 Villers-lès-Nancy Cedex, France
{Horatiu.Cirstea, Claude.Kirchner}@loria.fr

September 4, 1998

Abstract

The recently introduced ρ-calculus [Cirstea and Kirchner, 1998b] permits to express in a uniform and simple way first-order rewriting, λ-calculus and non-deterministic computations as well as their combination. In this work, we recall the main properties of ρ-calculus and we give a full first-order presentation of this rewriting calculus using an explicit substitution setting $\rho\sigma$ that generalizes the $\lambda\sigma$-calculus. Its basic properties in the untyped as well as typed cases are presented. We then detail how to use the ρ-calculus to give an operational semantics to the rewrite based language ELAN.

1 Introduction

We have introduced in [Cirstea and Kirchner, 1998b] the ρ-calculus as a general rewriting computation paradigm that uniformly integrates the first-order and higher-order computation paradigms.

Because these two paradigms are individually extremely attractive and have useful complementary features, their combination have attracted a lot of attention, especially in the last decade. This has been addressed either by enriching first-order rewriting with higher-order capabilities or by adding to λ-calculus algebraic features allowing one to deal with equality in an efficient way. In the first case, we find the works on CRS [Klop *et al.*, 1993] and other higher-order rewriting systems [Wolfram, 1993; Nipkow and Prehofer, 1998], in the second case the works on combination of λ-calculus with term rewriting [Okada, 1989; Breazu-Tannen, 1988; Gallier and Breazu-Tannen, 1989; Jouannaud and Okada, 1997] to mention only a few.

We come up with another point of view because of our previous works on the control of term rewriting [Kirchner *et al.*, 1995; Vittek, 1994; Borovanský *et al.*, 1998]. Indeed we realized that the tool needed to control rewriting should be made explicit and could be itself naturally described using rewriting. This leads to the ρ-calculus; the rewriting calculus which is introduced in [Cirstea and Kirchner, 1998b].

In this calculus the application of a rule (say $a \to b$) to the top level of a term (like the constant a) is represented as the object of the calculus $[a \to b](a)$ which evaluates to the singleton $\{b\}$. When the application of a rule fails like in $[a \to b](c)$, the expression evaluates to the empty set $\emptyset$. Of course variables may appear in rewrite rules like $[f(x) \to x](f(a))$ that evaluates to $\{a\}$. In fact when evaluating this expression, the variable x is bound to a by the matching mechanism and we have shown in [Borovanský *et al.*, 1998] how a rewrite rule can be considered as a function with matching as the parameter passing mechanism.

A λ-expression $\lambda x.t$ could be represented as the rule $x \to t$ and the β-reduction of the redex $(\lambda x.t\ u)$ is exactly the rewrite of the ρ-term $[x \to t](u)$ into the ρ-term $\{t\{x/u\}\}$. Of course, the substitution mechanism should preserve the correct variable bindings via the appropriate α-conversion. In order to make this point clear and as initiated in [Dowek *et al.*, 1995], we make a strong distinction between substitution (which takes care of variable binding) and grafting (that performs replacement directly). So basic ρ-calculus objects are built from the signature, the abstraction operator $\to$ and the application operator $[\ ](\ )$.

Another important feature of the ρ-calculus is its ability to handle non-determinism in the sense of sets of results. This is achieved via the explicit handling of reduction result sets, including the empty set that records failure of rule application. It allows us to make use in an explicit and direct way of non-terminating or non-confluent (equational) rewrite systems. For example if the symbol $+$ is assumed to be commutative then applying the rule $x + y \to x$ to the term $a + b$ results in $\{a, b\}$ since there are two different ways to apply this rule modulo commutativity, and it gives two different results.

After introducing the ρ-calculus and its main interest for the combination of first and higher-order computations, the purpose of this paper is to define and study a explicit substitution calculus for the ρ-calculus and to provide a full still simple semantics for the rewrite based language ELAN.

In the first part we shortly introduce the basic ρ-calculus, its syntax, deduction rules and examples of expressiveness. We then define and study a calculus of explicit substitutions, $\rho\sigma$, that generalizes the standard $\lambda\sigma$ calculus and for which a simple type system could be also defined. Finally we detail how to use the ρ-calculus to give an operational semantics to the ELAN language. This permits in particular to give a full account of ELAN's strategy objects and of their use. We conclude by providing research direc-

tions that are of main interest in the context of ELAN and more generally of rewrite based languages like ASF+SDF [van den Brand *et al.*, 1997], ML, Maude [Clavel *et al.*, 1998] or CafeOBJ [Futatsugi and Nakagawa, 1996].

This paper does not contain all the technical details and proofs that can be found in [Cirstea and Kirchner, 1998a].

2 The ρ-calculus

A calculus is defined by four components:

- First its syntax that makes precise the formation of the terms manipulated by the calculus as well as the substitutions that are used during the evaluation mechanism. In the case of ρ-calculus, the core of the term formation relies on rewrite rules and rule application.
- The description of the substitution application on terms. This description is often given at the meta-level, except for explicit substitution frameworks like the one we describe in a later part of this paper. In the case of ρ-calculus, substitutions are higher-order substitutions and not grafting.
- The matching algorithm used to bind variables to their actual arguments. In the case of ρ-calculus, this is in general higher-order matching and in practical cases pattern, equational or syntactic matching.
- The rules describing the way the calculus operates. It is the glue between the previous components and the simplicity and clarity of these rules are fundamental for the calculus usability.

This section makes explicit all these components for the ρ-calculus.

2.1 Syntax

Definition 2.1 We consider a set $\mathcal{F} = \bigcup_m \mathcal{F}_m$ of ranked function symbols, where $\mathcal{F}_m$ is the subset of function symbols of arity m, and $\mathcal{X}$ a set of variables. We denote by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ the set of first-order terms built on $\mathcal{F}$ using the variables in $\mathcal{X}$. The set of basic ρ-terms, denoted $\mathcal{R}(\mathcal{F}, \mathcal{X})$, can be inductively defined using a BNF syntax:

$$\text{terms} \quad t \quad ::= \quad x \mid \{t, \ldots, t\} \mid f(t, \ldots, t) \mid [t](t) \mid t \to t$$

where $x \in \mathcal{X}$ and $f \in \mathcal{F}$.

We adopt a very general discipline for the rule formation, and we do not enforce any of the standard restrictions like non-variable left-hand-sides or

occurrence of the right-hand-side variables in the left-hand-side. We also allow rules containing rules as well as rule application.

The main intuition behind this syntax is that a rewrite rule is an abstractor, the left-hand-side of which determines the bound variables and some contextual information. Having new variables in the right-hand-side is just the ability to have free variables in the calculus. We will come back to this later but to support the intuition of the reader let us mention that the λ-term $\lambda x.(y\ x)$ corresponds to the ρ-term $x \to [y](x)$ and that standard first-order rewrite rules [Dershowitz and Jouannaud, 1990; Baader and Nipkow, 1998] are clearly embedded in the calculus.

Example 2.1 Some more examples of ρ-terms are:

- $[x \to y](a)$; as expected, we will see later why the result is $\{y\}$.
- $[x \to (x \to x)]([x \to y](a))$; a more complicated ρ-term similar to the λ-term $((\lambda x.\lambda x.x)\ (\lambda x.y\ a))$.
- $[(a \to b) \to c](a)$; a more complicated ρ-term without corresponding λ-term.

As in λ-calculus, α-conversion should be used in order to obtain a correct substitution calculus and the first order substitution (called here grafting) is not directly suitable for our calculus.

Computing the matching substitutions from a ρ-term t to a ρ-term t' is the fundamental parameter in the evaluation rules of the ρ-calculus.

For a theory T a *T-match-equation* is a formula of the form $t \ll^?_T t'$, where t and t' are ρ-terms. A substitution σ is a solution of the T-match-equation $t \ll^?_T t'$ if $T \models \sigma(t) = t'$. A *T-matching system* is a conjunction of T-match-equations. A substitution is solution of a T-matching system P if it is a solution of all the T-match-equations in P. We denote by $\mathbb{F}$ a T-matching system without solution.

Since in general we consider higher-order terms as well as arbitrary equational theories, T-matching is in general undecidable, even when restricted to first-order equational theories [Jouannaud and Kirchner, 1991].

A T-matching system is called *trivial* when all substitutions are solution of it. We define the function *Solution* on a T-matching system $\mathcal{S}$ as returning the set of all T-matches of $\mathcal{S}$ when $\mathcal{S}$ is not trivial and $\{\mathbb{ID}\}$ where $\mathbb{ID}$ is the identity substitution when $\mathcal{S}$ is trivial. When the matching algorithm fails *Solution* returns the empty set ($\emptyset$).

One can think at using constraints as in constrained higher-order resolution [Huet, 1973] or constrained deduction [Kirchner *et al.*, 1990] if one wants to use undecidable matching theories. But we are primarily interested here in the decidable cases and we restrict ourselves later in this paper to just considering syntactic matching. In this case we assume that the left

members of matching equations are composed only of first-order terms (i.e. not containing sets, arrows or applications).

2.2 Deduction Rules

Given a rewrite rule $l \to r$, its application to a term t (denoted $[l \to r](t)$) consists of replacing the term t by σr, where σ is the substitution obtained by T-matching l on t and σr represents the application of the substitution on the term r as detailed above. The semantics of the application of a rewrite rule is given by the rules in Figure 1.

The $Fire$ rule *starts* the reduction process by T-matching the left-hand side of the rule on the argument of the rule. This generalizes the β-rule of λ-calculus. As we have seen before, the matching problem $l \ll^?_T t$ is solved in the theory T and depending on the theory T, we will obtain various rewriting calculi of interest.

The $Distrib, Batch, Switch$ rules and their $Clash$ versions describe the behavior of the application when one of the left-hand side, the right-hand side or the argument of the rule are (empty) sets.

The Set rule describes the semantics of a function with set arguments. The $Flat$ rule is used to flatten the sets of sets.

In order to push rule application deeper into terms, we introduce two $Congruence$ rules that deal with the application of a term of the form $f(t_1, \ldots, t_n)$ (where $f \in \mathcal{F}_n$) to another term of the same form.

It should be pointed out that the interpretation of f is overloaded: in the left-hand side of the rule $Congruence$ the first f operates on strategies and the second on terms; on the right-hand side the f operates on sets. Thus, we can say that for any symbol $f \in \mathcal{F}_n$ satisfying $f : A_1 \times \ldots \times A_n \to A$ we have two more symbols: $f_r : ((A_1 \to A_1) \times \ldots \times (A_n \to A_n)) \to (A \to A)$ which operates on strategies and $f_s : \{A_1\} \times \ldots \times \{A_n\} \to \{A\}$ which operates on sets.

The behavior of the "$\langle\langle\rangle\rangle$" operator is described by the rules in Figure 2. Note that these rules are applied at the meta-level and are not made explicit in the ρ-calculus. We are giving in next section a version of the ρ-calculus with explicit substitutions in the spirit of [Abadi *et al.*, 1991].

The $Propagate$ rule yields as a result the union of the partial results obtained by applying each of the substitutions to the respective term. When the matching problem has no solution the rule $Clash$ gives as result the empty set.

Notice that the last ρ-term presented in Example 2.1 does not satisfy this restriction.

Example 2.2 We present the reduction for a ρ-term which respects this restriction:

$Fire$	$[l \to r](t)$		
	$\Longrightarrow$		
	$r\langle\langle \mathcal{S}olution(l \ll^?_T t)\rangle\rangle$		
$Distrib$	$[\{s_1, \ldots, s_n\}](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^n [s_i](t)$		
$Batch$	$[s](\{t_1, \ldots, t_n\})$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^n [s](t_i)$		
$Switch_L$	$[\{l_1, \ldots, l_n\} \to r](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^n [l_i \to r](t)$		
$Switch_R$	$[l \to \{r_1, \ldots, r_n\}](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^n [l \to r_i](t)$		
Set	$f(A_1, \ldots, t, \ldots, A_n)$		
	$\Longrightarrow$		
	$\{f(a_1, \ldots, t, \ldots, a_n) \mid a_i \in A_i\}$		
$Congruence$	$[f(s_1, \ldots, s_n)](f(t_1, \ldots, t_n))$		
	$\Longrightarrow$		
	$f([s_1](t_1), \ldots, [s_n](t_n))$		
$Congruence\_fail$	$[f(s_1, \ldots, s_n)](g(t_1, \ldots, t_n))$	$\Longrightarrow$	$\emptyset$ if $f \neq g$
$Clash_D$	$[\emptyset](t)$	$\Longrightarrow$	$\emptyset$
$Clash_B$	$[t](\emptyset)$	$\Longrightarrow$	$\emptyset$
$Clash_{SL}$	$[\emptyset \to r](t)$	$\Longrightarrow$	$\emptyset$
$Clash_{SR}$	$[l \to \emptyset](t)$	$\Longrightarrow$	$\emptyset$
$Clash_S$	$f(\ldots, \emptyset, \ldots)$	$\Longrightarrow$	$\emptyset$
$Flat$	$\{\{t\}, \ldots\}$	$\Longrightarrow$	$\{t, \ldots\}$

Figure 1: Basic ρ_T-calculus

$Propagate$	$r\langle\langle\{s_1, \ldots, s_n\}\rangle\rangle$	$\rightsquigarrow$	$\{s_1(r), \ldots, s_n(r)\}$
$Clash$	$r\langle\langle\emptyset\rangle\rangle$	$\rightsquigarrow$	$\emptyset$

Figure 2: Substitution rules

$$[f(x,y) \to g(x,y)](f(a,b)) \stackrel{Fire}{\Longrightarrow} g(x,y)\langle\langle Solution(f(x,y) \ll^? f(a,b))\rangle\rangle$$
$$\Longrightarrow g(x,y)\langle\langle\{(x \mapsto a, y \mapsto b)\}\rangle\rangle \stackrel{Propagate}{\rightsquigarrow} \{g(a,b)\}$$

At this moment we can notice that the syntax and the deduction rules of the ρ-calculus can be restricted to some simpler calculi such as λ-calculus and term rewriting ([Cirstea and Kirchner, 1998b]). A λ-term of the form $\lambda\ \ x.t$ is represented by a ρ-term of type $x \to t$ and a rewrite rule is also a ρ rewrite rule.

3 The $\rho\sigma$-calculus

3.1 The Basic $\rho\sigma$-calculus

Up to now the application of substitutions was not part of the calculus. In order to make explicit the role of substitutions we define an explicit version of ρ-calculus. We start by giving the syntax and the basic deduction rules for an explicit $\rho\sigma$-calculus with explicit substitutions that will be extended in next sections with new syntactic elements and new rules.

3.2 Syntax

In what follows we concentrate on a presentation that relies on the de Bruijn's numbering for variables [de Bruijn, 1972]. This way the explicit renaming of variables is no longer needed, the renaming is described by an incrementation or decrementation of the integers representing variables.

If we consider a set $\mathcal{F} = \bigcup_m \mathcal{F}_m$ of ranked function symbols, where $\mathcal{F}_m$ is the subset of functions of arity m then the $\rho\sigma$-terms are formed according to the following rules:

- n ($n \in Nat$) is a term
- $\emptyset$ is a term
- if $t_1, \ldots, t_m$ are terms then $\{t_1, \ldots, t_m\}$ is a term
- if $t_1, \ldots, t_m$ are terms and $f \in \mathcal{F}_m$ then $f(t_1, \ldots, t_m)$ is a term
- if t and u are terms then $[t](u)$ is a term (application)
- if t and u are terms then $t \to_n u$ is a term (rewrite rule with n bounded variables)
- if t is a term and s is a substitution then $t\langle s\rangle$ is a term
- if t is a term and S is a set of substitutions then $t\langle\langle S\rangle\rangle$ is a term

with substitutions of the form:

- $\mathbb{ID}$ is a substitution ($identity$)
- $\uparrow$ is a substitution ($shift$)
- if s is a substitution then $\Uparrow(s)$ is a substitution ($lift$)
- if t is a term and s is a substitutions then $(t.s)$ is a substitution
- if s_1 and s_2 are substitutions then $(s_1 \circ s_2)$ is a substitution

We can express the same thing in BNF syntax:

terms	t	::=	$n \mid \emptyset \mid \{t,\ldots,t\} \mid f(t,\ldots,t)$ $\mid [t](t) \mid t \rightarrow_n t \mid t\langle s\rangle \mid t\langle\langle S\rangle\rangle$
substitutions	s	::=	$\mathbb{ID} \mid \uparrow \mid \Uparrow(s) \mid t.s \mid s \circ s$
set of substitutions	S	::=	$\emptyset \mid \{s,\ldots,s\}$

where $n \in Nat$ and $f \in \mathcal{F}$.

A rule of the form $(l \rightarrow_n r)$ represents a rule with n bounded variables by its left-hand side that is, l contains n free variables (e.g. $x+x+y \rightarrow_2 2x+y$).

3.3 Deduction Rules

The semantics of the application of a rewrite rule is given by the deduction rules of the $\rho\sigma_T$-calculus (Figure 3) and by the rules describing the application of a substitution on a term (Figure 4).

The deduction rules are similar to the ones of the ρ-calculus but this time we deal with indeces instead of variable names and the application of substitutions is explicit in the calculus.

The rule $Fire$ is the one that *starts* the reduction process by matching the left-hand side of the rewrite rule on the initial input term. The subsystem σ_ρ applies explicitly the obtained substitutions on the right-hand side of the rule.

As in the basic ρ-calculus the rules $Distrib$, $Batch$ and $Switch$ define the behavior of the reduction when some terms are sets. The Set rule describes the semantics of a function with set arguments as the set of functions with arguments belonging to these set arguments. The rules $Clash_D$, $Clash_B$, $Clash_{SL}$, $Clash_{SR}$ and $Clash_S$ describe the behavior of the empty set ($\emptyset$) on different situations, all of them giving as result the $\emptyset$ set.

In order to describe the application of a rule deeper into a term we introduce two $Congruence$ rules that deal with the application of a term of the form $f(s_1,\ldots,s_n)$ (where $f \in \mathcal{F}_n$) on another term with a function symbol as head symbol. If this latter symbol is f the substitutions s_i are propagated and if it is not the case the empty set is returned.

$Fire$	$[l \rightarrow_n r](t)$		
	$\Longrightarrow$		
	$r\langle\langle \mathcal{S}olution(l \ll^?_T t)\rangle\rangle$		
$Distrib$	$[\{s_1, \ldots, s_n\}](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^{n}[s_i](t)$		
$Batch$	$[s](\{t_1, \ldots, t_n\})$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^{n}[s](t_i)$		
$Switch_L$	$[\{l_1, \ldots, l_m\} \rightarrow_n r](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^{m}[l_i \rightarrow_n r](t)$		
$Switch_R$	$[l \rightarrow_n \{r_1, \ldots, r_m\}](t)$		
	$\Longrightarrow$		
	$\bigcup_{i=1}^{m}[l \rightarrow_n r_i](t)$		
Set	$f(A_1 \ldots, t, \ldots, A_n)$		
	$\Longrightarrow$		
	$\{f(a_1, \ldots, t, \ldots, a_n) \mid a_i \in A_i\}$		
$Congruence$	$[f(s_1, \ldots, s_n)](f(t_1, \ldots, t_n))$		
	$\Longrightarrow$		
	$f([s_1](t_1), \ldots, [s_n](t_n))$		
$Congruence\_fail$	$[f(s_1, \ldots, s_n)](g(t_1, \ldots, t_n))$	$\Longrightarrow$	$\emptyset$ if $f \neq g$
$Clash_D$	$[\emptyset](t)$	$\Longrightarrow$	$\emptyset$
$Clash_B$	$[s](\emptyset)$	$\Longrightarrow$	$\emptyset$
$Clash_{SL}$	$[\emptyset \rightarrow_n r](t)$	$\Longrightarrow$	$\emptyset$
$Clash_{SR}$	$[l \rightarrow_n \emptyset](t)$	$\Longrightarrow$	$\emptyset$
$Clash_S$	$f(t_1, \ldots, \emptyset, \ldots, t_n)$	$\Longrightarrow$	$\emptyset$
$Flat$	$\{\{t\}, \ldots\}$	$\Longrightarrow$	$\{t, \ldots\}$

Figure 3: Basic $\rho\sigma_T$-calculus

We consider a matching algorithm that returns a result of the form $nt_1.....nt_n.\mathbb{ID}$ if the classical matching algorithm ([Jouannaud and Kirchner, 1991]) returns the result $\{x_i \mapsto t_i\}_{i=1,...,n}$ for the same problem and nt_i are the de Bruijn representation of the terms t_i in the initial referential. The transformation of a term in the de Bruijn representation is described below. As in section 2.1 we define the function $\mathcal{S}olution$ on a T-matching system $\mathcal{S}$ as returning the set of all T-matches of $\mathcal{S}$ when $\mathcal{S}$ is not trivial, $\{\mathbb{ID}\}$ when $\mathcal{S}$ is trivial and the empty set ($\emptyset$) when the matching algorithm fails.

The rewriting rules of the reduction relation σ_ρ are very similar to the ones described in [Curien *et al.*, 1996] and [Pagano, 1998] for the relations σ and $\sigma_{\Uparrow}$ respectively.

We abbreviate the application n times of $\Uparrow (\Uparrow (\ldots (\Uparrow (s) \ldots)))$ by $\Uparrow^n (s)$.

The rules in σ_ρ applied on a term $t\langle\langle S\rangle\rangle$ yield a normal form that do not contain the application of a substitution if the substitution S does not represent a matching system without solution ($\emptyset$) and provide $\emptyset$ as solution in this latter case. The rules *set* and *empty* describe the application of a substitution on a set.

In the $\lambda\sigma$-calculus the term λa binds the variable 1 in the term a and we can say that λ has one (implicit) argument that is bound and that the binding arity of λ is (1). For a more detailed discussion on binding arities one should refer to [Pagano, 1998]. The unary symbol "λ" from the σ system is replaced by the binary symbol "$\rightarrow_n$" of a binding arity (n, n) (each of the two arguments of $\rightarrow_n$ contains n bounded variables). All the "f" symbols have a binding arity $(0,\ldots,0)$.

Definition 3.1 The calculus consisting of the deduction rules in Figure 3 and the subsystem σ_ρ (Figure 4) is called the $\rho\sigma$-calculus.

The transformation of a term with real names from the ρ-calculus in a term using natural indices is similar to the transformation of a λ term in a λ_{DB} term except the rule dealing with the rewrite rules.

We consider a list of bounded variables (e.g. $x.y.z.Nil$) that is called a referential. Having done a referential $\mathcal{R}$ we define recursively the translation of a term t, written $tr(t, \mathcal{R})$.

1. $tr(x, \mathcal{R}) = j$, where j is the first position of x in $\mathcal{R}$
2. $tr(f(t_1, \ldots, t_n, \mathcal{R})) = f(tr(t_1, \mathcal{R}), \ldots, tr(t_n, \mathcal{R}))$
3. $tr([a](b), \mathcal{R}) = [tr(a, \mathcal{R})](tr(b, \mathcal{R}))$
4. $tr(l \rightarrow r, \mathcal{R}) = tr(l, var(l).\mathcal{R}) \rightarrow_{\|var(l)\|} tr(r, var(l).\mathcal{R})$, with $\|var(l)\|$ representing the length of the list

where $var(t)$ is obtained by transforming the set of free variables, $FV(t)$ (section 2.1) into a list.

lam	$(l \to_n r)\langle s\rangle$	$\to$	$l\langle \Uparrow^n (s)\rangle \to_n r\langle \Uparrow^n (s)\rangle$
app	$[a](b)\langle s\rangle$	$\to$	$[a\langle s\rangle](b\langle s\rangle)$
$clos$	$a\langle s\rangle\langle t\rangle$	$\to$	$a\langle s \circ t\rangle$
$vs1$	$n\langle\uparrow\rangle$	$\to$	$(n+1)$
$vs2$	$n\langle\uparrow \circ s\rangle$	$\to$	$(n+1)\langle s\rangle$
fvc	$1\langle a.s\rangle$	$\to$	a
$fvl1$	$1\langle\Uparrow (s)\rangle$	$\to$	1
$fvl2$	$1\langle\Uparrow (s) \circ t\rangle$	$\to$	$1\langle t\rangle$
rvc	$(n+1)\langle a.s\rangle$	$\to$	$n\langle s\rangle$
$rvl1$	$(n+1)\langle\Uparrow (s)\rangle$	$\to$	$n\langle s\circ \uparrow\rangle$
$rvl2$	$(n+1)\langle\Uparrow (s) \circ t\rangle$	$\to$	$n\langle s \circ (\uparrow \circ t)\rangle$
id	$a\langle \mathbb{ID}\rangle$	$\to$	a
set	$\{a_1, \ldots, a_n\}\langle s\rangle$	$\to$	$\{a_1\langle s\rangle, \ldots, a_n\langle s\rangle\}$
$empty$	$\emptyset\langle s\rangle$	$\to$	$\emptyset$
ass	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$
map	$(a.s) \circ t$	$\to$	$a\langle t\rangle.(s \circ t)$
sc	$\uparrow \circ (a.s)$	$\to$	s
$sl1$	$\uparrow \circ \Uparrow (s)$	$\to$	$s\circ \uparrow$
$sl2$	$\uparrow \circ \Uparrow (s) \circ t$	$\to$	$s \circ (\uparrow \circ t)$
$l1$	$\Uparrow (s)\circ \Uparrow (t)$	$\to$	$\Uparrow (s \circ t)$
$l2$	$\Uparrow (s) \circ (\Uparrow (t) \circ u)$	$\to$	$\Uparrow (s \circ t) \circ u$
le	$\Uparrow (s) \circ (a.t)$	$\to$	$a.(s \circ t)$
il	$\mathbb{ID} \circ s$	$\to$	s
ir	$s \circ \mathbb{ID}$	$\to$	s
li	$\Uparrow (\mathbb{ID})$	$\to$	$\mathbb{ID}$
$f_{\Uparrow}$	$f(a_1, \ldots, a_n)\langle s\rangle$	$\to$	$f(a_1\langle s\rangle, \ldots, a_n\langle s\rangle)$
$base$	$a\langle\langle\{s_1, \ldots, s_n\}\rangle\rangle$	$\to$	$\{a\langle s_1\rangle, \ldots, a\langle s_n\rangle\}$
$fail$	$a\langle\langle\emptyset\rangle\rangle$	$\to$	$\emptyset$

Figure 4: The σ_ρ calculus

Example 3.1 If we consider the term $[f(x,y) \to g(x,y,z)](f(a,b))$ its transformation is $[f(1,2) \to_2 g(1,2,3)](f(a,b))$ in the referential $z.Nil$.

3.4 Properties of the $\rho\sigma$-calculus

The system applying substitutions is confluent and terminating as stated below:

Theorem 3.1 *σ_ρ is locally confluent and terminating.*

Proof: See [Cirstea and Kirchner, 1998a]. □

In order to prove now that $\rho\sigma$-calculus is itself confluent, we are using the notion of Explicit Reduction Systems(XRS) as defined by [Pagano, 1998].

Definition 3.2 *Explicit reduction systems (XRS)*
Let $\mathcal{F}$ be a signature, and $\mathcal{R}$ be a set of rewrite rules on $\mathcal{F}_{\Uparrow}$ (the term algebra of σ [Pagano, 1998]), such that all the rewrite rules $(l \to r)$ in $\mathcal{R}$ verify:

1. l is not a variable and all variables of r occur in l.
2. l is linear (a variable occur at most only one time).
3. l is a term build on the signature $\mathcal{F}_{\mathcal{X}}$ (terms built on the signature $\mathcal{F}$ and the set of term variables $\mathcal{X}$).

The term rewriting system defined on the term algebra $\mathcal{F}_{\Uparrow}$ and the rewriting relation $(\sigma \cup \mathcal{R})$ is said to be an *explicit reduction system* denoted XRS[$\mathcal{F}_{\Uparrow}$,$\mathcal{R}$].

Definition 3.3 A relation R is said *coherent* with σ (or σ-coherent) if and only if the following diagram can be closed:

$$\begin{array}{ccc} f & \xrightarrow{\mathcal{R}} & g \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle\sigma^*} \\ h & \xrightarrow[\sigma^*\,\mathcal{R}\,\sigma^*]{} & i \end{array}$$

The rules of such a rewrite system consist of first-order rules (denoted R^1) that by definitiondo not introduce substitutions and higher-order rules (denoted $R^>$) containing substitutions.

We consider now an XRS defined by a signature $\mathcal{F}$ and two sets of rewrite rules R^1 and $R^>$ such that:

1. $R^>$ and R^1 are coherent with σ,
2. $R^>$ is orthogonal and orthogonal with rules of R^1,

3. R^1 contains only left-linear FC-rules,

4. R^1 is confluent.

Theorem 3.2 *[Pagano, 1998] Under these hypotheses, the explicit reduction system XRS[$\mathcal{F}_{\Uparrow}$,$R^1 \cup R^>$] is confluent.*

We are proving the confluence of $\rho\sigma$-calculus by checking that $\rho\sigma$-calculus is an XRS satisfying the above conditions [Cirstea and Kirchner, 1998a]. This allows us to state:

Theorem 3.3 *We suppose that the Fire rule is applied when none of the other rules is applicable and the left-hand side, right-hand side and argument of the rule are completely normalized. Following this strategy, the $\rho\sigma$-calculus is confluent.*

Having proved the confluence of the $\rho\sigma$-calculus we would expect to have the same result for the ρ-calculus. For this we should proceed either by proving directly that the ρ-calculus is confluent or by establishing the relationship between the implicit and the explicit calculi. We take the latter approach which leads to the main result.

Theorem 3.4 *Given t and t' two ρ-terms. Then $t \stackrel{\rho}{\Longrightarrow} t'$ iff $tr(t, \mathcal{R}) \stackrel{\rho\sigma}{\Longrightarrow} tr(t', \mathcal{R})$ for any $\mathcal{R}$. If t and t' are α-equivalent then $tr(t, \mathcal{R})$ and $tr(t', \mathcal{R})$ are equal for any $\mathcal{R}$.*

Corollary 3.1 We suppose that the *Fire* rule is applied *when none* of the other rules is applicable and the left-hand side, right-hand side and argument of the rule are completely normalized. Following this strategy, the ρ-calculus is confluent.

3.5 Typed $\rho\sigma$-calculus

In this section we move to the typed $\rho\sigma$-calculus by introducing types in terms and substitutions. Contexts are lists of types and they are used to record the types of free variables. A context $A_1.A_2.\ldots.A_n$ associates the type A_i to the index i.

The syntax of the simply typed ρ-calculus is:

Types	$A ::=$	$K \mid A \to B \mid \{A, B, \ldots\}$
Contexts	$E ::=$	$nil \mid A.E$
Terms	$t ::=$	$n \mid \emptyset \mid f(t, \ldots, t) \mid \{t, \ldots, t\}$ $\mid [t](t) \mid t : A \to_n t \mid t\langle s \rangle \mid t\langle\langle S \rangle\rangle$
Substitutions	$s ::=$	$\mathbb{ID} \mid \uparrow \mid \Uparrow(s) \mid t.s \mid s \circ s$
Substitution Sets	$S ::=$	$\emptyset \mid \{s, \ldots, s\}$

Var	$A, E \vdash 1 : A$
Var_n	$\frac{E \vdash n : B}{A.E \vdash (n+1) : B}$
$Lambda\_n$	$\frac{A_1.\ldots.A_n.E \vdash a : A, A_1.\ldots.A_n.E \vdash b : B}{E \vdash (a : A \rightarrow_n b) : A \rightarrow B}$
App	$\frac{E \vdash b : A \rightarrow B, E \vdash a : A}{E \vdash [b](a) : \{B\}}$
$Clos$	$\frac{E \vdash s \triangleright E', E' \vdash a : A}{E \vdash a\langle s \rangle : A}$
$Decompose$	$\frac{E \vdash s_1 \triangleright E_1, E_1 \vdash a : A_1, \ldots}{E \vdash a\langle\langle\{s_1, \ldots, s_n\}\rangle\rangle : \{A_1, \ldots, A_n\}}$
Set	$\frac{E \vdash a_1 : A_1, \ldots, E \vdash a_n : A_n}{E \vdash f(a_1, \ldots, a_n) : B}$ $if(f : A_1 \times \ldots \times A_n \rightarrow_n B)$
Ens	$\frac{E \vdash a : A, E \vdash b : B}{E \vdash \{a, b\} : \{A, B\}}$
$Empty$	$E \vdash \emptyset : A$
Id	$E \vdash \mathbb{ID} \triangleright E$
$Failure$	$E \vdash \emptyset \triangleright F$
$Shift$	$A.E \vdash \uparrow \triangleright E$
$Lift$	$\frac{E \vdash s \triangleright E'}{A.E \vdash \Uparrow(s) \triangleright A.E'}$
$Cons$	$\frac{E \vdash a : A, E \vdash s \triangleright E'}{E \vdash a : A.s \triangleright A.E'}$
$Comp$	$\frac{E \vdash s'' \triangleright E'', E'' \vdash s' \triangleright E'}{E \vdash s' \circ s'' \triangleright E'}$

Figure 5: Typing rules

We divide the typing rules in two groups, one for giving types to terms, and one for giving environments to substitutions (Figure 5).

The rules $Fire$ and lam are enriched with typing information and the other rules remain unchanged.

$$Fire \quad [l : A \to_n r](t) \quad \to_n \quad \{r\langle\langle l \ll^?_T t\rangle\rangle\}$$

$$lam \quad (l : A \to_n r)\langle s\rangle \quad \to_n \quad l\langle \Uparrow^n (s)\rangle : A \to_n r\langle \Uparrow^n (s)\rangle$$

In the $\rho\sigma$-calculus there are some terms that do not have a normal form (the calculus is infinite). For example, we can consider the classical term $((\lambda x.xx)\lambda x.xx)$:

Example 3.2 $[1 \to_1 [1](1)](1 \to_1 [1](1))$
reduces to $[1](1)\langle\langle\{(1 \to_1 [1](1)).\mathbb{ID}\}\rangle\rangle$
$\{[1](1)\langle(1 \to_1 [1](1)).\mathbb{ID}\rangle\}$
$\{[1 \to_1 [1](1)](1 \to_1 [1](1))\}$
...
$\{\ldots\{[1 \to_1 [1](1)](1 \to_1 [1](1))\}\ldots\}$

In the typed $\rho\sigma$-calculus the corresponding term:
$[1 : A \to_1 [1](1)](1 : A \to_1 [1](1))$ is not well-typed.

Lemma 3.1 If a term a reduces to a' in $\rho\sigma$-calculus and $E \vdash a : A$, then $E \vdash a' : A$. If a substitution s reduces to s' in $\rho\sigma$-calculus and $E \vdash s \triangleright E'$, then $E \vdash s' \triangleright E'$.

3.6 Primal strategies

The basic ρ-calculus is already quite powerful because of the matching and substitution mechanisms it relies on, but we would like to enrich it further to allow an easy specification of choice operators. Indeed the whole calculus is intended to facilitate the specification and execution of non-deterministic possibly non-confluent rewrite systems as needed for concurrent systems as well as general deduction systems like theorem provers or constraint solvers.

The basic calculus could be enriched in two ways. The first one is to consider a fixpoint operator, the second one consists of an extention of the syntax and the inference rules of the calculus in order to integrate directly the desirable operators. We have chosen the second approach as it is more direct and close to what is done in ELAN as we will discuss later.

Considering the ρ-terms as functions that apply rewrite rules, we called them *strategies* and since we are building an extention of the basic terms, we call *primal strategies* basic terms to which we add a sequence operator in order to denote the successive application of two strategies.

To this end, the syntax from section 2.1 is extended with the operators **id** and **fail** that correspond respectively to the identity rule $(x \to x)$ and the failure rule whose application fails on any term $(x \to \emptyset)$.

terms $t ::= id \mid fail$

and the deduction rules are:

$Identity \quad id \implies x \to x$

$Fail \quad fail \implies x \to \emptyset$

We should add the corresponding rules for the σ_ρ:

$idp \quad id\langle s\rangle \to id$

$failp \quad fail\langle s\rangle \to fail$

It can be easily proved that the σ_ρ system is still confluent and terminating and the $\rho\sigma$ calculus is still confluent.

The sequential application of two strategies is achieved by using the concatenation operator ";":

terms $t ::= t; t$

The behavior of the concatenation operator is expressed by the following deduction rule:

$Compose \quad [s_1; s_2](t) \implies [s_2]([s_1](t))$

The corresponding rule added to σ_ρ is:

$comp \quad (s_1; s_2)\langle s\rangle \to s_1\langle s\rangle; s_2\langle s\rangle$

Having added these new operators the σ_ρ system remains confluent and terminating and the $\rho\sigma$ calculus is still confluent.

Definition 3.4 *Primal strategies* are built starting from rewrite rules $(l \to r)$ and the two operators id and $fail$ and using concatenation. Their application is defined by the rules in Figure 1, $Identity$, $Fail$ and $Compose$.

Example 3.3 If we consider the primal strategy

$$g(f(f(x) \to x)); g(f(x) \to x); g(a) \to a$$

then the only term on which the application of the strategy yields as result a non-empty set is $g(f(f(a)))$.

If the term is of the form $g(f(t))$ with t not of the form $f(t')$ then the $Congruence$ rule can be applied but the matching would fail and the result is the empty set. If t is of the form $f(t')$ with t' different from a then the application of the strategy $g(f(f(x) \to x)); g(f(x) \to x)$ to the term would yield as result $g(t')$ and we obtain once again the $\emptyset$ set due to the rule $Match\_Fail$. In the other cases the rule $Congruence\_fail$ would lead immediately to an empty set result.

The typing information are added for the new introduced symbols id, $fail$ and ";".

$Identity \quad E \vdash id : A \to A$

$Fail \quad E \vdash fail : A \to A$

$$Compose \quad \frac{E \vdash s_1 : A \to B, E \vdash s_2 : B \to C}{E \vdash s_1; s_2 : A \to C}$$

3.7 Elementary strategies

The strategy language is now enriched by operators allowing us to describe the computation space in a convenient way.

The added operators select the final result(s) from a list of results and choose the strategy to be applied from a list of strategies. For the first type of selection we introduce the unary operators **one** and **all** that operate on strategies and whose semantics are described by the following rules:

$$\begin{array}{lll} One & [one(s)](t) \implies & \{t'\} \\ & & \text{where } t' \in [s](t) \\ All & [all(s)](t) \implies & [s](t) \end{array}$$

Then we can also introduce selection operators **select-one**, **select-first**, **select-all** on strategy tuples ([Borovanský *et al.*, 1998]). Their semantics is defined by the rules:

$$\begin{array}{ll} Select\_first & [select\_first(s_1,\ldots,s_n)](t) \implies [s_j](t) \\ & \qquad \text{if } \bigcup_{i=1}^{j-1}[s_i](t) = \emptyset \text{ and } [s_j](t) \neq \emptyset \\ Select\_one & [select\_one(s_1,\ldots,s_n)](t) \implies [s_i](t) \\ & \qquad \text{if } [s_i](t) \neq \emptyset \\ Select\_all & [select\_all(s_1,\ldots,s_n)](t) \implies \bigcup_{i=1}^{n}[s_i](t) \end{array}$$

The operator **select-first** returns the result of the first successful application, **select-one** selects one of the successful applications and **select-all** provides the union of all (successful) applications. These selection operators allow us to describe the operators **first**, **dc**(dont care) and **dk**(dont know) by the rules:

$$\begin{array}{lll} Dk & [dk(s_1,\ldots,s_n)](t) \implies & [select\_all(all(s_1),\ldots,all(s_n))](t) \\ Dc & [dc(s_1,\ldots,s_n)](t) \implies & [select\_one(all(s_1),\ldots,all(s_n))](t) \\ First & [first(s_1,\ldots,s_n)](t) \implies & [select\_first(all(s_1),\ldots,all(s_n))](t) \end{array}$$

Intuitively, the selectors are used to select the strategy and the **one** and **all** operators determine the number of results to be returned when applying the respective strategy to the input term.

Some other strategies can be constructed starting from the above operators. For example, an elementary strategy that chooses one result in each set of results provided by a sequence of strategies is :

$$One \quad [one(s_1,\ldots,s_n)](t) \implies [select\_one(one(s_1),\ldots,one(s_n))](t)$$

Note also that the result selector can be made more precise when a different representation for results is chosen. For example, when using lists, the **one** operator could be completed with "nth" ones.

3.8 Defined strategies

Using this kind of operators and primal strategies we can define more elaborate strategies. For example, a function **map** can be defined in an explicit way by:

$Map_1 \quad [map(s)](nil) \implies nil$
$Map_2 \quad [map(s)](a.l) \implies [s](a).[map(s)](l)$

or in an implicit (and more compact) way:

$Map \quad map(s) \implies first(nil,\ s.map(s))$

Other basic strategy definitions like **iterate** or **repeat** can be given in a similar way.

More complicated control structures can be constructed if we extend the rule *Congruence* with new cases that deal with constant head symbols in the strategy application:

$$\begin{array}{lll} Congruence\_M & [\Phi(s_1,\ldots,s_n)](f(t_1,\ldots,t_n)) & \implies \\ & f([s_1](t_1),\ldots,[s_n](t_n)) & \\ Congruence\_S & [\Psi(s)](f(t_1,\ldots,t_n)) & \implies \\ & f([s](t_1),\ldots,[s](t_n)) & \end{array}$$

where Φ, Ψ are constants of the language ($\Phi, \Psi \notin \mathcal{F}$) acting as wildcards for the head symbol of a strategy application.

A strategy operator applying a strategy in an *innermost* way is then easy to define:

$Im \quad im(s) \implies \Psi(im(s)); s$

while the *outermost* strategy has a similar description:

$Om \quad om(s) \implies s; \Psi(om(s))$

Depending on the order used in evaluating the arguments of the function in the right-hand side of the rule $Congruence\_S$ the strategy im is a *leftmost innermost, rightmost innermost* or *random innermost* strategy.

4 The ELAN environment and its ρ-calculus semantics

4.1 ELAN's rewrite rules

ELAN 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 and to experiment with for example the combination of theorem provers, constraint solvers and decision procedures [Vittek, 1994; Kirchner *et al.*, 1995; Borovanský *et al.*, 1996]. It has been experimented on several non-trivial applications ranging from constraint solvers to logic programming and automated theorem proving[1].

ELAN provides a kernel that implements the leftmost innermost standard rewriting strategy, the elementary strategies, and which allows the iteration of the construction on defined strategies as defined in the previous section.

[1] see `http://www.loria.fr/equipes/protheo/PROJECTS/ELAN/elan.html` for more details

A partial semantics could be given to an ELAN program using rewriting logic [Meseguer, 1992], but more conveniently ELAN's rules can be expressed using the ρ-calculus and thus an ELAN program is just a set of ρ-terms. A rule with no conditions and no local assignments $l \Rightarrow r$ is represented by $l \rightarrow r$. The local assignments are let-like constructions that allow applications of strategies on some terms. The BNF syntax of an ELAN rule is:

<rule> ::= [[*<label>*]] *<term>* => *<term>* { *<local evaluation>* } *

<local evaluation> ::= **if** *<boolean term>*
| **where** *<variable>* := ([*<strategy>*]) *<term>*

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

Example 4.1 An ELAN rule is:

```
[AND] H |-  (P && Q)          =>  True
                        where S1 := (dedstrat) H |- Q
                        if S1 = True
                        where S2 := (dedstrat) H |- P
                        if S2 = True                    end
```

In ELAN the local assignments and conditions are evaluated sequentially in textual order. If the strategies used in the local assignments are terminating any ELAN rule can be expressed as a rule containing the list of local assignments from the initial rule followed by the condition representing the conjunction of all the conditions of the initial rule. If one of the strategies used in the rule is not terminating but the local assignment containing it is evaluated only after an unsatisfied condition the application of the rule would yield a (set of) result(s) while the corresponding rule with all the conditions at the end does not terminate due to the non-terminating local strategy that is not guarded by the condition.

So, obviously, the two rules are not operationally equivalent. The transformed rules always evaluate all the assignments even if this is not the case for the initial rule due to some unsatisfied conditions that stops the evaluation immediately.

Example 4.2 The transformation of the ELAN rule from the previous example is:

```
[AND] H |-  (P && Q)          =>  True
                        where S1 := (dedstrat) H |- Q
                        where S2 := (dedstrat) H |- P
                        if S1 = True  and  S2 = True      end
```

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

4.2 The ρ-calculus representation of ELAN rules

As noted before term rewriting can be described using the ρ-calculus. Conditional rewrite rules can also be represented in a simple extension of the ρ-calculus. The syntax is extended with a conditional operator **if** and with the boolean constants $True$ and $False$:

terms $t ::= if\ t \mid True \mid False$

The following rules describe the semantics of the **if** operator:

$If\!e \quad [if\ \{c_1, \ldots, c_n\}](t) \Longrightarrow \bigcup_{i=1}^{n} [if\ c_i](t)$

$If\!t \quad [if\ True](t) \Longrightarrow \{t\}$

$If\!f \quad [if\ False](t) \Longrightarrow \emptyset$

For the last two rules we can give equivalent (implicit) definitions:

$If\!t \quad if\ True \Longrightarrow (x \to x)$

$If\!f \quad if\ False \Longrightarrow (x \to \emptyset)$

The semantics of the conditions should be described by a set of rules consisting in the equality operator case of:

$Eq \quad a = a \Longrightarrow True$

$NEq \quad a = b \Longrightarrow False$, if $a \neq b$

Using these considerations a conditional rewrite rule of the form $(l \to r\ \ if\ \ c)$ is represented by the ρ-term $(l \to [if\ \ c](r))$ and an appropriate ρ-term represents any derivation:

Proposition 4.1 [Cirstea and Kirchner, 1998a] Given t and t' two terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathcal{R}$ a first order conditional term rewrite system. If $t \xrightarrow{*}_{\mathcal{R}} t'$ then there exists a ρ-term u constructed using the primal strategies of $\mathcal{R}$ such that $[u](t) \xrightarrow{*}_{\rho} \{t'\}$.

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: $l \to_n r$, where n is the number of variables in l. More complicated ELAN rewrite rules can be given as well a ρ-calculus representation like in the following example:

Example 4.3 The ELAN's rule

[deriveSum] $p_1 + p_2 \Rightarrow p'_1 + p'_2$
where $p'_1 := (derive)p_1$
where $p'_2 := (derive)p_2$

is represented by the ρ-term $p_1 + p_2 \Rightarrow [derive]p_1 + [derive]p_2$.

If we consider more general ELAN rules containing local assignments and conditions on the local variables some further extensions of the original ρ-calculus are needed. A new operator is used for dealing with these conditions and three rules describing the semantics of the **IF** operator are introduced. The last two ones are applied only when the arguments are completely normalized and the first one is not applicable:

$$
\begin{array}{lll}
IFset & IF(s_1,\ldots,s_n,N,\{t_1,\ldots,t_m\}) & \Longrightarrow \\
 & \bigcup_{i=1}^{m} IF(s_1,\ldots,s_n,N,t_i) & \\
IFsing & IF(s,N,t) & \Longrightarrow \\
 & [if\ [N](t)]([s](t)) & \\
IFnorm & f(IF(s_1,\ldots,s_n,N,t)) & \Longrightarrow \\
 & [if\ [N](t)](f([s_1](t),\ldots,[s_n](t)) &
\end{array}
$$

The rule $IFset$ is used when we have a set of conditions to be evaluated and provides as result the set of all terms constructed starting from the set of conditions. The rule $IFnorm$ is applied when we do not have a set of conditions but only one condition.

We denote by $t\lceil s_1\rceil^{p_1}\ldots\lceil s_n\rceil^{p_n}$ or $t\lceil s_1,\ldots,s_n\rceil^{p_1,\ldots,p_n}$ the term$t$ containing the subterms $s_1,\ldots,s_n$ at the positions $p_1,\ldots,p_n$ respectively.

In general we consider an ELAN rule that has the following form:

$$
\begin{array}{llll}
\text{[label]} & l & \Rightarrow & r\lceil x_1\rceil^{q_1}\ldots\lceil x_n\rceil^{q_n} \\
 & & & \text{where } x_1 := (s_1)t_1 \\
 & & & \ldots \\
 & & & \text{where } x_n := (s_n)t_n \\
 & & & \text{if } C\lceil x_1\rceil^{p_1}\ldots\lceil x_n\rceil^{p_n}
\end{array}
$$

where t_{i+1} can depend on x_i. For a term t whose variables are $\{x_1,\ldots,x_n\}$ we define the n projection rules as:

$$
\begin{array}{llll}
pr_1 & C & \rightarrow & x_1 \\
 & & & \ldots \\
pr_n & C & \rightarrow & x_n
\end{array}
$$

Some additional auxiliary rules can be considered in order to represent some of the subterms of the right-hand side of the rule that are constructed starting from these projections.

We denote $normalize$ the ρ-term corresponding to the leftmost innermost strategy of ELAN. Then, the ELAN rule above is expressed as the ρ-term:

$l \rightarrow r\lceil IF(ps_1,\ldots,ps_n,normalize,C\lceil[s_1](t_1)\rceil^{p_1}\ldots\lceil[s_n](t_n)\rceil^{p_n})\rceil^{q_1,\ldots,q_n}$

where ps_i are constructed using pr_i and eventually some of the auxiliary rules.

The way this transformation applies on a rewrite rule and the corresponding reduction of the obtained ρ-term is illustrated by:

Example 4.4

$$
\begin{array}{llll}
\text{For the following rewrite rule } x & \Rightarrow & h(z,l(y)) & \\
 & & \text{where } y := (s1)\ x+1 & \\
 & & \text{where } z := (s2)\ x+1 & \\
 & & \text{if } y > z &
\end{array}
$$

the projection rules are:

$pr1 \quad (y > z \rightarrow z)$

$pr2 \quad (y > z \to y)$
and the auxiliary rule:
$aux \quad (u \to l(u))$
and the transformed term is:
$x \to h(IF(pr1, pr2; aux, normalize, [s1](x+1) > [s2](x+1)))$

This term applied to the term 1 with $s_1 = s_2 = dk(2 \to 3, 2 \to 4)$ yields the following derivation:

$[x \to h(IF(pr1, pr2; aux, normalize, [s1](x+1) > [s2](x+1))](1)$
$\Rightarrow \quad \{h(IF(pr1, pr2; aux, normalize, [s1](2) > [s2](2)))\}$
$\Rightarrow \quad \{h(IF(pr1, pr2; aux, normalize, \{3,4\} > \{3,4\}))\}$
$\Rightarrow \quad \{h(IF(pr1, pr2; aux, normalize, 3 > 3)) \cup$
$h(IF(pr1, pr2; aux, normalize, 3 > 4)) \cup$
$h(IF(pr1, pr2; aux, normalize, 4 > 3)) \cup$
$h(IF(pr1, pr2; aux, normalize, 4 > 4))\}$
$\Rightarrow \quad \{[if\ [normalize](3 > 3)](h([pr1](3 > 3), [pr2; aux](3 > 3))) \cup$
$[if\ [normalize](3 > 4)](h([pr1](3 > 4), [pr2; aux](3 > 4))) \cup$
$[if\ [normalize](4 > 3)](h([pr1](4 > 3), [pr2; aux](4 > 3))) \cup$
$[if\ [normalize](4 > 4)](h([pr1](4 > 4), [pr2; aux](4 > 4))) \cup \}$
$\Rightarrow \quad \{[if\ False](h([pr1](3 > 3), [pr2; aux](3 > 3))) \cup$
$[if\ False](h(\ldots)) \cup$
$[if\ True](h([pr1](4 > 3), [pr2; aux](4 > 3))) \cup$
$[if\ False](h(\ldots))\}$
$\Rightarrow \quad \{\emptyset \cup \ \emptyset \cup \ \{h([y > z \to y](4 > 3), [y > z \to z; u \to l(u)](4 > 3))\} \cup \ \emptyset\}$
$\Rightarrow \quad \{\{h(\{3\}, \{l(4)\})\}\}$
$\Rightarrow \quad \{\{\{h(3, l(4))\}\}\}$
$\Rightarrow \quad \{h(3, l(4))\}$

The semantics of the primal and elementary strategies of ELAN are reflected by the corresponding strategies in the ρ-calculus and the user defined strategies are similar to the ρ defined strategies.

5 Conclusion

We have presented the general properties of the ρ-calculus and given a first order presentation of the calculus using explicit substitutions in the un-typed as well as typed cases.

The ρ-calculus is both conceptually simple as well as very expressive. This allowed us to show how the ρ-calculus can be used to give a semantics to ELAN rules. More generally, the applications to many other frameworks have to be investigated, including rewrite based languages like ASF+SDF, ML, Maude or CafeOBJ but also non-deterministic transition systems.

Among the many research directions concerned by the use of the ρ-calculus for the combination of first-order and higher-order paradigms we are

now deepening the relationship of ρ-calculus with rewriting logic [Meseguer, 1992] and we plan to investigate the relationship of this calculus with higher-order rewrite concepts like CRS and HOR [Oostrom and Raamsdonk, 1993].

References

[Abadi *et al.*, 1991] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Baader and Nipkow, 1998] Franz Baader and Tobias Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

[Borovanský *et al.*, 1996] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Marian Vittek. ELAN: A logical framework based on computational systems. In José Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[Borovanský *et al.*, 1998] Peter Borovanský, Claude Kirchner, and Hélène Kirchner. A functional view of rewriting and strategies for a semantics of ELAN. In Masahiko Sato and Yoshihito Toyama, editors, *The Third Fuji International Symposium on Functional and Logic Programming*, pages 143–167, Kyoto, April 1998. World Scientific. Also report LORIA 98-R-165.

[Breazu-Tannen, 1988] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 82–90, 1988.

[Cirstea and Kirchner, 1998a] Horatiu Cirstea and Claude Kirchner. *Combining Higher-Order and First-Order Computation Using ρ-calculus: Towards a semantics of* ELAN. *Full Paper.* http://www.loria.fr/~cirstea/Papers/RhoCaculus.ps, 1998.

[Cirstea and Kirchner, 1998b] Horatiu Cirstea and Claude Kirchner. ρ-calculus. Its syntax and basic properties. Research report 98-R-218, LORIA, August 1998.

[Clavel *et al.*, 1998] M. Clavel, F. Durn, S. Eker, P. Lincoln, and J. Meseguer. An Introduction to Maude (Beta Version). Technical report, SRI International, Computer Science Laboratory, Menlo Park, (CA, USA), March 1998.

[Curien *et al.*, 1996] P.-L. Curien, Th. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[de Bruijn, 1972] N. G. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, 75(5):381–392, 1972.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Dowek *et al.*, 1995] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order unification via explicit substitutions, extended abstract. In Dexter Kozen, editor, *Proceedings of LICS'95*, pages 366–374, San Diego, June 1995.

[Futatsugi and Nakagawa, 1996] K. Futatsugi and A. Nakagawa. An Overview of Cafe Project. In *Proceedings of First CafeOBJ Workshop*, Yokohama (Japan), August 1996.

[Gallier and Breazu-Tannen, 1989] J. Gallier and V. Breazu-Tannen. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *16th Colloquium Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150. Springer-Verlag, 1989.

[Huet, 1973] G. Huet. A mechanization of type theory. In *Proceeding of the third international joint conference on artificial intelligence*, pages 139–146, 1973.

[Jouannaud and Kirchner, 1991] J.-P. Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[Jouannaud and Okada, 1997] Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 28 February 1997.

[Kirchner *et al.*, 1990] Claude Kirchner, Hélène Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[Kirchner *et al.*, 1995] Claude Kirchner, Hélène Kirchner, and Marian 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.

[Klop *et al.*, 1993] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

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

[Nipkow and Prehofer, 1998] Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations.* Kluwer, 1998.

[Okada, 1989] Mitsuhiro Okada. Strong normalizability for the combined system of the typed λ calculus and an arbitrary convergent term rewrite system. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation: ISSAC '89 / July 17–19, 1989, Portland, Oregon*, pages 357–363, New York, NY 10036, USA, 1989. ACM Press.

[Oostrom and Raamsdonk, 1993] Vincent van Oostrom and Femke fan 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.

[Pagano, 1998] Bruno Pagano. X.R.S : Explicit Reduction Systems - A First-Order Calculus for Higher-Order Calculi. In Claude Kirchner and Hélène Kirchner, editors, *15th International Conference on Automated Deduction*, LNAI 1421, pages 72–87, Lindau, Germany, July 5–July 10, 1998. Springer-Verlag.

[van den Brand *et al.*, 1997] M. van den Brand, P. Olivier, L. Moonen, and T. Kuipers. Implementation of a Prototype for the New ASF Meta-environment. In *Proceedings of International Workshop on Theory and Practice of Algebraic Specifications ASF+SDF 97, Amsterdam (The Nederlands)*, Workshops in Computing. Springer-Verlag, September 1997.

[Vittek, 1994] Marian 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.

[Wolfram, 1993] D. A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1993.