Using Rewriting and Strategies for Describing the B Predicate Prover

Horatiu Cirstea, Claude Kirchner

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
54600 Villers-lès-Nancy Cedex, France
{Horatiu.Cirstea, Claude.Kirchner}@loria.fr

Abstract

The ρ-calculus 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 paper, we present the main components of the ρ-calculus and we give a full first-order presentation of this rewriting calculus using an explicit substitution setting, called $\rho\sigma$, that generalizes the $\lambda\sigma$-calculus. The basic properties of the non-explicit and explicit substitution versions are presented. We then detail how to use the ρ-calculus to give an operational semantics to the rewrite rules of the ELAN language.

1 Introduction

The ρ-calculus is a calculus of explicit rewrite rule application that uniformly integrates the first-order and higher-order computation paradigms.

Because these two paradigms are individually extremely attractive and have useful complementary features, a lot of attention has been devoted to the study of their combination, 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 [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 a calculus of explicit rule application.

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 $[a \to b](c)$ because a does not match the constant 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 reduction 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.

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 an explicit substitution calculus for the ρ-calculus and to provide a full still simple semantics for the rewrite rules used in the ELAN language.

In Section 2 we first introduce the syntax of the basic ρ-calculus and we give examples of its expressiveness. The deduction rules are then presented together with considerations on the matching mechanism and on the application of the substitutions. At the end of this section we explain the strategy guiding the deduction rules that should be used in order to obtain a confluent calculus. In Section 3, we define and study a calculus of explicit substitutions, $\rho\sigma$, that generalizes the standard $\lambda\sigma$ calculus. The syntax and the deduction rules are presented in this new context and the same confluence results as in the non-explicit case are found. In Section 4 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 directions 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 several proofs that can be found in [Cirstea and Kirchner, 1998].

2 The ρ-calculus

The ρ-calculus is defined by five 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 [Miller, 1991], equational [Jouannaud and Kirchner, 1991] 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.
- The strategy guiding the application of the rules. Depending on the strategy employed we can obtain different properties for the calculus.

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

2.1 Syntax, substitution and matching

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 $\mathcal{F}_\rho$, can be inductively defined by the following grammar:

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

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 used when defining the rewriting relation 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. We consider that the symbols $\{\}$ and $\emptyset$ both represent the empty set.

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, in order 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 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 an important 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 $\mathbf{F}$ 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). Notice that the last ρ-term presented in Example 2.1 does not satisfy this restriction.

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 are obtaining various rewriting calculi of interest.

Since rule application occurs at the top level of terms, 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. When we have the same head symbol for the two terms of the application $[u](v)$ the arguments of the term u are applied on those of the term v argumentwise. If the head symbols are not similar an empty set is obtained.

The $Distrib, Batch$ and $Switch_L, Switch_R$ rules describe the propagation of the sets on the application and abstraction operators. The $OpOnSet$ rule describes the semantics of a function with set arguments. The $Flat$ rule is used to flatten the sets of sets.

The behavior of the "$\langle\!\langle\rangle\!\rangle$" operator is described by the rule *Propagate* in Figure 2. Note that this rule is applied at the meta-level and is not made explicit in the ρ-calculus. We are giving in the 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 result is the empty set.

Example 2.2 The reduction for a ρ-term representing the application of a rewrite rule on a term:

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

Fire		$[l \to r](p)$
	$\Longrightarrow$	$r\langle\langle \mathcal{S}olution(l \ll^?_T p)\rangle\rangle$
Congruence		$[f(u_1,\ldots,u_n)](f(v_1,\ldots,v_n))$
	$\Longrightarrow$	$\{f([u_1](v_1),\ldots,[u_n](v_n))\}$
Congruence_fail		$[f(u_1,\ldots,u_n)](g(v_1,\ldots,v_m))$
	$\Longrightarrow$	$\emptyset$
Distrib		$[\{u_1,\ldots,u_n\}](v)$
	$\Longrightarrow$	$\{[u_1](v),\ldots,[u_n](v)\}$
Batch		$[v](\{u_1,\ldots,u_n\})$
	$\Longrightarrow$	$\{[v](u_1),\ldots,[v](u_n)\}$
$\mathbf{Switch_L}$		$\{u_1,\ldots,u_n\} \to v$
	$\Longrightarrow$	$\{u_1 \to v,\ldots,u_n \to v\}$
$\mathbf{Switch_R}$		$u \to \{v_1,\ldots,v_n\}$
	$\Longrightarrow$	$\{u \to v_n,\ldots,u \to v_n\}$
OpOnSet		$f(v_1,\ldots,\{u_1,\ldots,u_m\},\ldots,v_n)$
	$\Longrightarrow$	$\{f(v_1,\ldots,u_1,\ldots,v_n),\ldots,f(v_1,\ldots,u_m,\ldots,v_m)\}$
Flat		$\{u_1,\ldots,\{v_1,\ldots,v_n\},\ldots,u_m\}$
	$\Longrightarrow$	$\{u_1,\ldots,v_1\ldots,v_n,\ldots,u_m\}$

Figure 1: Basic ρ_T-calculus

Propagate $\quad r\langle\langle\{\sigma_1,\ldots,\sigma_n\}\rangle\rangle \quad \rightsquigarrow \quad \{\sigma_1\ r,\ldots,\sigma_n\ r\}$

Figure 2: Substitution rules

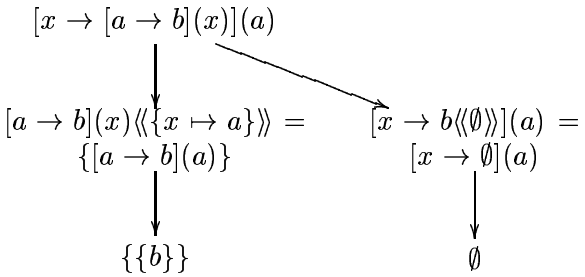
At this moment we can notice that the syntax and the deduction rules of the ρ-calculus can be restricted to some simpler calculi such as λ-calculus and term rewriting. A λ-term of the form $\lambda x.t$ is represented by a ρ-term of type $x \to t$ and a rewrite rule is also a ρ-rewrite-rule (see Section 4.2).

2.3 Properties of the ρ-calculus

The general ρ-calculus is obviously non-confluent. The main reasons for the non-confluence are undesired matching failures involving uninstantiated variables and unreduced terms and the set representation for the results of reductions.

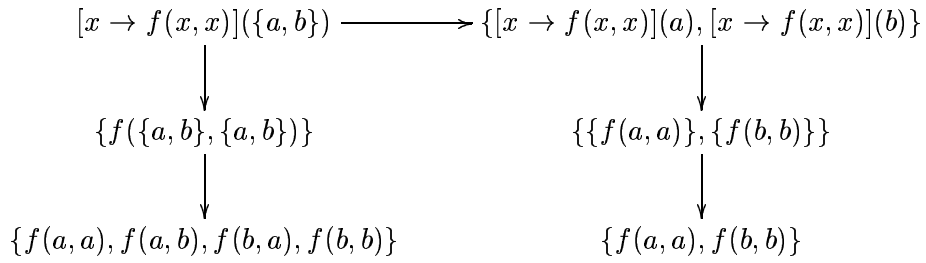
The first main reason comes from the fact that a matching system containing free variables or unreduced terms that lead to a failure can become successful when the variables are instantiated or the terms reduced. Example 2.3 illustrates the case of a matching failure due to an uninstantiated variable. For the case of unreduced terms we can consider the term $[a \to b]([a \to a](a))$ that reduces to $\{b\}$ or $\emptyset$ following the position where the rule *Fire* is applied first. Similarly, we can obtain different derivations for terms containing non-left linear rules ($[f(x,x) \to b](f(a,[a \to a](a)))$) or when a rule is applied on a set ($[f(x) \to b](\{f(a)\})$).

Example 2.3



As far as it concerns the handling of the non-determinism (sets of results) we have two types of problems. On one hand, we can have empty sets representing a matching failure that are eventually not propagated properly (for instance $[x \to b](\emptyset)$ reduces to $\{b\}$ or $\emptyset$ according to the rule applied) and on the other hand, we have sets with more than one element that can lead to undesirable results in a non-linear context (Example 2.4).

Example 2.4



Every time a ρ-term is reduced using the rules $Fire$, $Congruence$ and $Congruence\_fail$ of the ρ-calculus, a set is generated. These rules are the ones that describe the application of a rule at the top level or deeper in a term. The set obtained when applying one of the above rules can trigger the application of the other rules of the calculus. These rules deal with the (propagation of) sets and compute a "set-normal form" for the ρ-terms by pushing out the set braces and flattening the imbricated sets.

Therefore, we consider that the rules of the ρ-calculus consist of a set of *deduction* rules ($Fire$, $Congruence$, $Congruence\_fail$) and a set of *computation* rules ($Distrib$, $Batch$, $Switch_L$, $Switch_R$, $OpOnSet$, $Flat$) and that the reduction behaves as a deduction modulo ([Dowek *et al.*, 1998]).

Let us now define the relations induced by the rules of the ρ-calculus and present their main properties.

Definition 2.2 We call Set the reduction relation over $\mathcal{F}_\rho$ induced by the rules $Distrib$, $Batch$, $Switch_L$, $Switch_R$, $OpOnSet$ and $Flat$.

The following relations are induced by the relation Set:

- $\longrightarrow_S$ is the one step Set-reduction (the compatible closure of Set),
- $\stackrel{*}{\longrightarrow}_S$ is the Set-reduction (the transitive-reflexive closure of $\longrightarrow_S$),
- $=\!=_S$ is the equivalence relation generated by $\stackrel{*}{\longrightarrow}_S$.

Definition 2.3 We call $Fire$ the reduction relation over $\mathcal{F}_\rho$ induced by the rules $Fire$, $Congruence$ and $Congruence\_fail$. Starting from this relation, we consider the following relations induced by the relations $Fire$ and Set:

- $\longrightarrow_F$ is the compatible closure of the relation $Fire$,
- $\longrightarrow_{F/S}$ is the relation $\longrightarrow_F$ modulo the relation $=\!=_S$ defined as usual ([Aho *et al.*, 1972]): given two ρ-terms u, v we have $u \longrightarrow_{F/S} v$ when there exist two terms u', v' such that $u =\!=_S u'$ and $u' \longrightarrow_F v'$ and $u =\!=_S u'$,
- $\stackrel{*}{\longrightarrow}_{F/S}$ is the transitive-reflexive closure of the relation $\longrightarrow_{F/S}$.

Note that the relation $\stackrel{*}{\longrightarrow}_{F/S}$ is not the transitive closure of the relation $\longrightarrow_F$ ($\stackrel{*}{\longrightarrow}_F$) modulo the relation $=\!=_S$.

Since we can consider the reduction in the ρ-calculus as the relation induced by the rules of the calculus ($\longrightarrow_F \cup \longrightarrow_S$) or as a rewriting modulo ($\longrightarrow_{F/S}$) we are proving the confluence of the two relations.

As already said the calculus is not confluent if no strategy is used for guiding the application of the reduction rules but the confluence can be obtained if a reduction strategy is imposed.

Definition 2.4 We call *ConfStrat* the strategy whose reductions are denoted by $\longrightarrow_{\mathcal{CS}}$ and that satisfies the next conditions, where the notation $u\lceil t\rceil_p$ denotes the term u having t at occurrence p.

- If $u\lceil t\rceil_p \xrightarrow{*}_{\mathcal{CS}} u\lceil\emptyset\rceil_p$ then for all term v such that $u\lceil t\rceil_p \xrightarrow{*}_{\mathcal{CS}} v$, we have $v \xrightarrow{*}_{\mathcal{CS}} \emptyset$ (i.e. "error" propagation is strict),
- if $[l \to r](p) \xrightarrow{Fire}_{\mathcal{CS}} r'\lceil p\rceil_i\lceil p\rceil_j$ then for all term v such that $p \xrightarrow{*}_{\mathcal{CS}} v$, there exists a ρ-term t such that $v \xrightarrow{*}_{\mathcal{CS}} \{t\}$ or $v \xrightarrow{*}_{\mathcal{CS}} \emptyset$ (i.e. v cannot be reduced to a set with more than one element).

From now on we consider that the rules of the calculus are guided by the strategy *ConfStrat*. Of course, more operational strategies can be given for obtaining the confluence of the calculus.

For proving the confluence of $\longrightarrow_{F/S}$ we try to find a relation δ that has the diamond property and whose transitive closure is the relation $\xrightarrow{*}_{F/S}$. In this case, we can conclude that the relation $\xrightarrow{*}_{F/S}$ has the diamond property and thus, $\longrightarrow_{F/S}$ is confluent.

We can choose for δ the reflexive closure of $\longrightarrow_{F/S}$. Unfortunately, it can be easily checked that this relation does not have the diamond property. We shall take an approach similar to the "parallel reduction" due to *Tait and Martin-Löf* and prove that $\longrightarrow_{F\|/S}$ is confluent and its transitive closure is $\xrightarrow{*}_{F/S}$.

Lemma 2.1 If $\longrightarrow_{F\|}$ is the parallelization of the relation $\longrightarrow_F$ then $\longrightarrow_{F\|}$ is strongly confluent.

Proof: By induction on the structure of the ρ-terms and using a "substitution lemma". □

Lemma 2.2 The relation $\longrightarrow_S$ is confluent and terminating.

Proof: We prove the termination by using a polynomial interpretation for the rewrite rules. The local confluence is obtained by proving the convergence of all critical pairs. Thus, we can conclude the confluence of the relation $\longrightarrow_S$. □

Proposition 2.1 The relation $\longrightarrow_{F/S}$ is confluent.

Proof: First, we show that if $t \longrightarrow_S t'$ and $t \longrightarrow_{F\|} s$, then there exists s' such that $t' \xrightarrow{*}_S \longrightarrow_{F\|} \xrightarrow{*}_S s'$ and $s \xrightarrow{*}_S s'$. The confluence of the relation $\longrightarrow_S$ and the strong confluence of the relation $\longrightarrow_{F\|}$ allow us to prove that $\longrightarrow_{F\|/S}$ has the diamond property.

It is then easy to prove that the relation $\xrightarrow{*}_{F/S}$ is the transitive closure of the relation $\longrightarrow_{F||/S}$ and thus, we conclude that $\longrightarrow_{F/S}$ is confluent. □

It is worth mentioning that the confluence of the relation induced by the rules of the calculus is similar to the approach proposed in [Curien *et al.*, 1996] for proving the confluence of $\lambda_{\Uparrow}$.

Proposition 2.2 The relation $\xrightarrow{*}_S \longrightarrow_F \xrightarrow{*}_S$ is confluent.

Proof: The Lemma of Yokouchi ([Curien *et al.*, 1996]) is used to show the strong confluence of $\xrightarrow{*}_S \longrightarrow_{F||} \xrightarrow{*}_S$. Since $\longrightarrow_F \subset \longrightarrow_{F||} \subset \xrightarrow{*}_F$ we obtain the confluence of $\xrightarrow{*}_S \longrightarrow_F \xrightarrow{*}_S$. □

3 The ρ_σ-calculus

Up to now the substitution application 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 the ρ_σ-calculus with explicit substitutions and we present the confluence results.

Comparing to the $\lambda_{\Uparrow}$-calculus one should notice the ability to have several variables bound by the ρ-abstractor ($\rightarrow$) and the possibility to use contextual information handled by the matching mechanism.

3.1 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 renaming of variables is handled via 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:

terms $\quad t ::= n \mid \emptyset \mid \{t, \ldots, t\} \mid f(t, \ldots, t) \mid [t](t) \mid t \rightarrow_n t \mid t\langle s \rangle$

substitutions $\quad s ::= \mathbb{ID} \mid \uparrow \mid \Uparrow(s) \mid t.s \mid s \circ s$

where $n \in \mathbf{N}^*$ and $f \in \mathcal{F}$.

A rule of the form $(l \rightarrow_n r)$ represents a rule with n bound variables by its left-hand side that is, l contains n free variables (e.g. $1+1+2 \rightarrow_2 1+2$).

3.2 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 indices instead of variable names and the application of substitutions is explicit in the calculus.

The rule *Fire starts* the reduction process by matching the left-hand side of the rewrite rule on the initial input term and applying the obtained substitution(s) on the right-hand side of the rule. The implicit $\langle\langle\rangle\rangle$ substitution mechanism becomes explicit in the new form of the rule $Fire$. The sub-system σ_ρ applies explicitly the obtained substitutions on the right-hand side of the rule.

All the other rules are very similar to their non-explicit counterparts.

We consider a matching algorithm that returns a result of the form $nt_1 \ldots . nt_n.\mathbb{ID}$ if the classical matching algorithm ([Jouannaud and Kirchner, 1991]) returns the result $\{x_i \mapsto t_i\}_{i=1,\ldots,n}$ for the same problem and nt_i are the de Bruijn representation of the terms t_i in the corresponding 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 by $\uparrow^n$ the composition of n symbols $\uparrow$ (i.e. $\uparrow \circ \ldots \circ \uparrow$) and by $\Uparrow^n (s)$ the application n times of $\Uparrow$ (i.e. $\Uparrow (\ldots (\Uparrow (s) \ldots))$).

The rules in σ_ρ applied on a term $t\langle s\rangle$ 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 application of a substitution on an abstraction, application, set and function is described by the rules lam, app, set and $f_{\Uparrow}$ respectively.

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 about binding arities one should refer to [Pagano, 1998]. The unary symbol "λ" from the σ system is replaced in our calculus by the binary symbol "$\rightarrow_n$" of a binding arity (n, n) (each of the two arguments of $\rightarrow_n$ contains n bound variables). All the first-order symbols "f" have a binding arity (0,...,0).

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

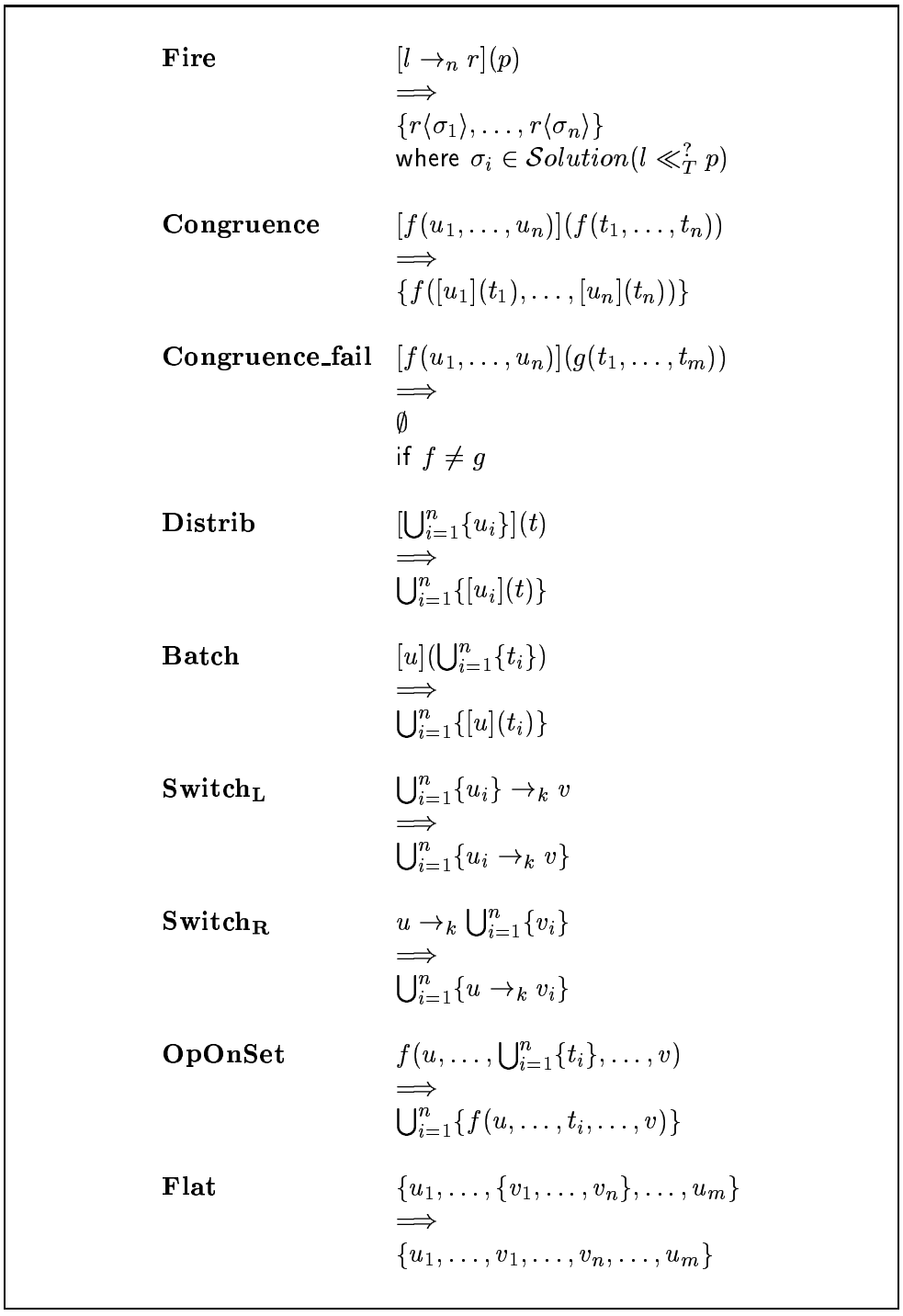


Figure 3: Basic $\rho\sigma_T$-calculus

lam	$(u \to_n v)\langle s\rangle$	$\Rightarrow$	$u\langle \Uparrow^n (s)\rangle \to_n v\langle \Uparrow^n (s)\rangle$
app	$[u](v)\langle s\rangle$	$\Rightarrow$	$[u\langle s\rangle](v\langle s\rangle)$
clos	$u\langle s\rangle\langle t\rangle$	$\Rightarrow$	$u\langle s \circ t\rangle$
vs1	$n\langle\uparrow\rangle$	$\Rightarrow$	$(n+1)$
vs2	$n\langle\uparrow \circ s\rangle$	$\Rightarrow$	$(n+1)\langle s\rangle$
fvc	$1\langle u.s\rangle$	$\Rightarrow$	u
fvl1	$1\langle\Uparrow (s)\rangle$	$\Rightarrow$	1
fvl2	$1\langle\Uparrow (s) \circ t\rangle$	$\Rightarrow$	$1\langle t\rangle$
rvc	$(n+1)\langle u.s\rangle$	$\Rightarrow$	$n\langle s\rangle$
rvl1	$(n+1)\langle\Uparrow (s)\rangle$	$\Rightarrow$	$n\langle s \circ\uparrow\rangle$
rvl2	$(n+1)\langle\Uparrow (s) \circ t\rangle$	$\Rightarrow$	$n\langle s \circ (\uparrow \circ t)\rangle$
id	$u\langle\mathbb{ID}\rangle$	$\Rightarrow$	u
set	$\bigcup_{i=1}^{n}\{u_i\}\langle s\rangle$	$\Rightarrow$	$\bigcup_{i=1}^{n}\{u_i\langle s\rangle\}$
ass	$(s \circ t) \circ v$	$\Rightarrow$	$s \circ (t \circ v)$
map	$(u.s) \circ t$	$\Rightarrow$	$u\langle t\rangle.(s \circ t)$
sc	$\uparrow \circ (u.s)$	$\Rightarrow$	s
sl1	$\uparrow \circ \Uparrow (s)$	$\Rightarrow$	$s \circ \uparrow$
sl2	$\uparrow \circ \Uparrow (s) \circ t$	$\Rightarrow$	$s \circ (\uparrow \circ t)$
l1	$\Uparrow (s) \circ \Uparrow (t)$	$\Rightarrow$	$\Uparrow (s \circ t)$
l2	$\Uparrow (s) \circ (\Uparrow (t) \circ v)$	$\Rightarrow$	$\Uparrow (s \circ t) \circ v$
le	$\Uparrow (s) \circ (u.t)$	$\Rightarrow$	$u.(s \circ t)$
il	$\mathbb{ID} \circ s$	$\Rightarrow$	s
ir	$s \circ \mathbb{ID}$	$\Rightarrow$	s
li	$\Uparrow (\mathbb{ID})$	$\Rightarrow$	$\mathbb{ID}$
$\mathbf{f}_{\Uparrow}$	$f(u_1, \ldots, u_n)\langle s\rangle$	$\Rightarrow$	$f(u_1\langle s\rangle, \ldots, u_n\langle s\rangle)$

Figure 4: The σ_ρ 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 bound 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. $tr(l \to r, \mathcal{R}) = tr(l, var(l).\mathcal{R}) \to_{\|var(l)\|} tr(r, var(l).\mathcal{R})$,

where $var(l)$ represents the list of free variables of the term l and $\|var(l)\|$ its length.

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 $z.Nil$.

3.3 Properties of the ρ_σ-calculus

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

Lemma 3.1 σ_ρ is locally confluent and terminating.

Proof: We use the methods proposed in [Curien *et al.*, 1996] and [Pagano, 1998] for the relations σ and $\sigma_{\Uparrow}$ respectively. □

In order to prove now that the ρ_σ-calculus is itself confluent, we can use the notion of Explicit Reduction Systems(XRS) as defined by [Pagano, 1998] or a direct proof as done for $\lambda_{\Uparrow}$ in [Curien *et al.*, 1996].

The first approach consists in expressing the ρ_σ-calculus as an XRS satisfying the confluence conditions stated in [Pagano, 1998]. The most restrictive condition is the orthogonality between the high-order rules ($Fire$ in our case) and the first-order rules (all the other rules presented in Figure 3). When an appropriate strategy for guiding the application of the rule $Fire$ is used, this condition is satisfied and the calculus is confluent:

Proposition 3.1 We suppose that the $Fire$ rule is applied on a sub-term $[l \to_n r](p)$ of a term t *only when* r and p contain no redexes and no sets and p contains no free variables that are bound in t. Following this strategy, the ρ_σ-calculus is confluent.

The definitions from Section 2.3 are extended for the relations induced by the rules of the ρ_σ-calculus and the same conditions on the application of the rule *Fire* are considered. For the direct proof of the confluence of the calculus, when the strategy from the Definition 2.4 is used, we consider two appropriate relations and we show that the conditions for the Lemma of Yokouchi ([Curien *et al.*, 1996]) are satisfied. The first relation is $\longrightarrow_{F\|}$ and the second one is $(\xrightarrow{*}_\sigma \longrightarrow_S \xrightarrow{*}_\sigma)$ and, for simplicity, we denote this last one by $\longrightarrow_{\sigma^* S \sigma^*}$.

Lemma 3.2 The relation $\longrightarrow_{\sigma^* S \sigma^*}$ is confluent and terminating.

Proof: The termination of $\longrightarrow_{\sigma^* S \sigma^*}$ is proved by giving a polynomial interpretation to the operators of the calculus.

First, we show the "compatibility" between $\longrightarrow_S$ and $\longrightarrow_\sigma$:
if $t \longrightarrow_S t'$ and $t \longrightarrow_\sigma s$ then it exists s' such that $t' \xrightarrow{*}_\sigma s'$ and $s \xrightarrow{*}_\sigma \longrightarrow_S \xrightarrow{*}_\sigma s'$. Using the compatibility between the two relations and their confluence we obtain the confluence of $\longrightarrow_{\sigma^* S \sigma^*}$ by induction on the number of $\longrightarrow_\sigma$ reductions. □

Proposition 3.2 The ρ_σ-calculus is confluent.

Proof: We show that the relations $\longrightarrow_{F\|}$ and $\longrightarrow_{\sigma^* S \sigma^*}$ are compatible in the sense explained in the previous lemma and we apply the Lemma of Yokouchi ([Curien *et al.*, 1996]) for the relations $\longrightarrow_{F\|}$ and $\longrightarrow_{\sigma^* S \sigma^*}$. □

As one would expect, we have a strong relationship between the ρ-calculus and the ρ_σ-calculus:

Proposition 3.3 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 $\mathcal{R}$.

3.4 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 extension of the syntax and of 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 extension of the existing syntax of 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 $id \quad \Longrightarrow \quad x \to x$

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

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 $[s_1; s_2](t) \quad \Longrightarrow \quad [s_2]([s_1](t))$

Definition 3.2 *Primal strategies* are built starting from rewrite rules (i.e. ρ-terms of the form $l \to r$) and the two operators id and $fail$ and using concatenation. Their application is defined by the rules in Figure 1, *Identity*, $Fail$ and $Compose$.

One can notice that the operators introduced in this section are just shortcuts for some particular ρ-terms and thus, the properties of the ρ-calculus are still valid for the calculus extended with the above aliases.

Example 3.2 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 empty set due to a matching failure when applying the last rule. In the other cases the rule $Congruence\_fail$ would lead immediately to an empty set result.

3.5 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:

One $[one(s)](t) \implies \{t'\}$ where $t' \in [s](t)$

All $[all(s)](t) \implies [s](t)$

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:

Select_first $[select\_first(s_1, \ldots, s_n)](t) \implies [s_j](t)$ if $\bigcup_{i=1}^{j-1}[s_i](t) = \emptyset$ and $[s_j](t) \neq \emptyset$

Select_one $[select\_one(s_1, \ldots, s_n)](t) \implies [s_i](t)$ if $[s_i](t) \neq \emptyset$

Select_all $[select\_all(s_1, \ldots, s_n)](t) \implies \bigcup_{i=1}^{n}[s_i](t)$

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:

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 $[first(s_1, \ldots, s_n)](t) \implies [select\_first(all(s_1), \ldots, all(s_n))](t)$

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 :

Dc_One $[dc\_one(s_1, \ldots, s_n)](t) \implies [select\_one(one(s_1), \ldots, one(s_n))](t)$

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.

It is easy to show that the **dk** operator can be expressed in the basic ρ-calculus. It is an open problem whether this is the case for the **dc** and **first** operators.

3.6 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:

Map1 $[map(s)](nil) \implies nil$
Map2 $[map(s)](a.l) \implies [s](a).[map(s)](l)$

or in an implicit (and more compact) way:

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

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 special head symbols in the strategy application:

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 $[\Psi(s)](f(t_1,\ldots,t_n)) \implies f([s](t_1),\ldots,[s](t_n))$

where Φ, Ψ are operators of the language ($\Phi, \Psi \notin \mathcal{F}$) acting as wildcards for the head symbol of a strategy application. These operators are similar to the ones presented in [Visser and el Abidine Benaissa, 1998].

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

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

while the *outermost* strategy has a similar description:

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

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 presented in the previous section.

[1] see `http://www.loria.fr/ELAN/` 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 list of ρ-terms. A rule with no conditions and no local assignments $l \Rightarrow r$ is represented by $l \to r$. The local assignments are let-like constructions that allow applications of strategies on some terms. The general syntax of an ELAN rule is:

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

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 rule (strategy).

Example 4.1 An example of an ELAN rule describing one of the deduction rules of the sequent calculus is:

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

A conditional rewrite rule of the form

$l \to r \ \ if \ \ c$

is represented by the ρ-term

$l \to [\{True \to r, False \to \emptyset\}]([normalize](c))$

or even the simpler (but maybe less suggestive) one:

$l \to [True \to r]([normalize](c))$

where *normalize* represents the strategy used to evaluate the condition.

Example 4.2 We consider the rewrite rule $f(x) \to g(x) \ \ if(x \geq 1)$ applied to the term $f(2)$.

$[f(x) \to [True \to g(x)]([normalize](x \geq 1))](f(2)) \Longrightarrow$
$\{[True \to g(2)]([normalize](2 \geq 1))\} \Longrightarrow \{[True \to g(2)](\{True\})\} \Longrightarrow$
$\{\{[True \to g(2)](True)\}\} \Longrightarrow \{\{\{g(2)\}\}\} \Longrightarrow \{g(2)\}$

Indeed, an appropriate ρ-term represents any derivation:

Proposition 4.1 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 $l \to r$ and a conditional rule is expressed as above. The ELAN rewrite rules with local assignments 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 => p_1' + p_2'
                         where p_1' := (derive)p_1
                         where p_2' := (derive)p_2
```

can be represented by one of the following two ρ-terms

$p_1 + p_2 \to [derive](p_1) + [derive](p_2)$

$p_1 + p_2 \to [p_1' \to [p_2' \to p_1' + p_2'][derive](p_2))][derive](p_1))$

If we consider more general ELAN rules containing local assignments as well as conditions on the local variables, the combination of the previous two methods should be done carefully. If we had used the former representation for this kind of rules we would have obtained some incorrect results like in the Example 4.4. In order to simplify the terms considered we omit the strategies used for reducing the conditions and simply denote $[normalize](c)$ by c.

Example 4.4 We consider the following ELAN rule:

```
[] x => y
        where y := (dk(1 => 2,1 => 3)) x
        if y >= 3 end
```

that would be represented by the ρ-term

$x \to [True \to [dk(1 \to 2, 1 \to 3)](x)]([dk(1 \to 2, 1 \to 3)](x) >= 3)$

that applied to the term 1 yields the following derivation:

$[x \to [True \to [dk(1 \to 2, 1 \to 3)](x)]([dk(1 \to 2, 1 \to 3)](x) >= 3)](1)$

$\Longrightarrow \{[True \to [dk(1 \to 2, 1 \to 3)](1)]([dk(1 \to 2, 1 \to 3)](1) >= 3)\}$

$\Longrightarrow \{[True \to \{2, 3\}](\{2, 3\} >= 3)\}$

$\Longrightarrow \{[True \to \{2, 3\}](2 >= 3) \cup [True \to \{2, 3\}](3 >= 3)\}$

$\Longrightarrow \{[True \to \{2, 3\}](False) \cup [True \to \{2, 3\}](True)\}$

$\Longrightarrow \{\emptyset \cup [True \to \{2, 3\}](True)\}$

$\Longrightarrow \{2, 3\}$

while the correct result is: $\{3\}$

The problem in the Example 4.4 is the double evaluation of the local variable y: once in the condition and once in the right-hand side of the rule. If the local variable is evaluated to a set of results this set is returned if one of its elements satisfies the condition while the appropriate result would be the subset of elements satisfying the condition. Thus, we need a mechanism that evaluates only once all the local assignments in a rule.

We denote by $t\lceil s_1\rceil_{p_1} \dots \lceil s_n\rceil_{p_n}$ the term t containing the sub-terms $s_1, \dots, s_n$ at the positions $p_1, \dots, p_n$ respectively.

Without loosing generality, we consider that an ELAN rule that has the following form:

$$\begin{array}{llll} \text{[label]} & l & \Rightarrow & r\lceil x_1\rceil_{q_1}\lceil x_2\rceil_{q_2} \\ & & & \text{where } x_1 := (s_1)t_1 \\ & & & \text{if } C_1\lceil x_1\rceil_{p_1} \\ & & & \text{where } x_2 := (s_2)t_2 \\ & & & \text{if } C_2\lceil x_1\rceil_{r_1}\lceil x_2\rceil_{r_2} \end{array}$$

where t_{i+1} can depend on x_i.

We denote by *normalize* the ρ-term corresponding to the leftmost innermost strategy of ELAN. If we use an approach similar to the latter ρ-representation of conditional rules, the ELAN rule above is expressed as the ρ-term:

$$\begin{array}{llllll} l \to & [x_1 \to & [\{True \to & [x_2 \to [\{ & True \to r\lceil x_1\rceil_{q_1}\lceil x_2\rceil_{q_2}, & \\ & & & & False \to \emptyset\}](C_2\lceil x_1\rceil_{r_1}\lceil x_2\rceil_{r_2}) & \\ & & &]([s_2](t_2)), & & \\ & & False \to \emptyset\}](C_1\lceil x_1\rceil_{p_1}) & & & \\ &]([s_1](t_1)) & & & & \end{array}$$

or the simpler one:

$$\begin{array}{llllll} l \to & [x_1 \to & [True \to & [x_2 \to & [True \to r\lceil x_1\rceil_{q_1}\lceil x_2\rceil_{q_2}] & \\ & & & & (C_2\lceil x_1\rceil_{r_1}\lceil x_2\rceil_{r_2}) & \\ & & &]([s_2](t_2))] & & \\ & & (C_1\lceil x_1\rceil_{p_1}) & & & \\ &]([s_1](t_1)) & & & & \end{array}$$

We notice that, when evaluating the application of such a rule, the ρ-rule *Fire* should be applied on top before applying it on the right-hand side of the rule, that is exactly the condition requested to get a confluent ρ-calculus.

The way this transformation applies on a rewrite rule and the corresponding reduction of the obtained ρ-term is illustrated in Example 4.5.

Example 4.5 For the following rewrite rule

```
[]x => h(z,l(y))
        where y:= (s1) x+1
        where z:= (s2) x+1
        if y > z
```

the transformed term is:

$$x \to [y \to [z \to [True \to h(z, l(y))](y > z)]([s_2](x+1))]([s_1](x+1))$$

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

$$
\begin{array}{ll}
 & [x \to [y \to [z \to [True \to h(z, l(y))](y > z)]([s_2](x+1))]([s_1](x+1))](1) \\
\Rightarrow & \{[y \to [z \to [True \to h(z, l(y))](y > z)]([s_2](1+1))]([s_1](1+1))\} \\
\Rightarrow & \{[y \to [z \to [True \to h(z, l(y))](y > z)]([s_2](1+1))](\{3, 4\})\} \\
\ldots & \\
\Rightarrow & \{\{\{\emptyset\} \cup \{\emptyset\}\} \cup \{\{\{h(3, l(4))\}\} \cup \{\emptyset\}\}\} \\
\Rightarrow & \{\{\{\{h(3, l(4))\}\}\}\} \\
\Rightarrow & \{h(3, l(4))\}
\end{array}
$$

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.

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.

[Aho *et al.*, 1972] A. V. Aho, R. Sethi, and J. D. Ullman. Code optimization and finite Church-Rosser systems. In *Proceedings of Courant Computer Science*, pages 89–105. Prentice Hall, 1972.

[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, 1998] 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.

[Clavel *et al.*, 1998] M. Clavel, F. Durán, 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.

[Dowek *et al.*, 1998] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. Rapport de Recherche 3400, Institut National de Recherche en Informatique et en Automatique, April 1998. `http://pauillac.inria.fr/~dowek/RR-3400.ps.gz`.

[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.

[Miller, 1991] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming: International Workshop, Tübingen, Germany, December 1989*, volume 475 of *Lecture Notes in Computer Science*, pages 253–281. Springer-Verlag, 1991.

[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.

[Visser and el Abidine Benaissa, 1998] Eelco Visser and Zine el Abidine Benaissa. A core language for rewriting. In Claude Kirchner and Hélène 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.

[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.