Rewriting and Multisets in the Rewriting Calculus and ELAN

Horatiu Cirstea, Claude Kirchner

Rewriting and Multisets in ρ-calculus and ELAN

Horatiu Cirstea & Claude Kirchner
LORIA and INRIA and UHP
615, rue du Jardin Botanique
54600 Villers-lès-Nancy Cedex, France
{Horatiu.Cirstea,Claude.Kirchner}@loria.fr

Abstract

The ρ-calculus is a new calculus that integrates in a uniform and simple setting first-order rewriting, λ-calculus and non-deterministic computations. The main design concept of the ρ-calculus is to make all the basic ingredients of rewriting explicit objects, in particular the notions of rule *application* and *multisets of results*. This paper describes the calculus from its syntax to its basic properties in the untyped case. The ρ-calculus embeds first-order conditional rewriting and λ-calculus and it can be used in order to give an operational semantics to the rewrite based language ELAN. We show how the set-like data structures are easily represented in ELAN and how this can be used in order to specify the Needham-Schroeder public-key protocol.
Keywords: Rewriting, Strategy, Multisets, Matching.

1 Introduction

It is a common claim that rewriting is ubiquitous in computer science and mathematical logic. And indeed the rewriting concept appears from the very theoretical settings to the very practical implementations. Some extreme examples are the mail system under Unix that uses rules in order to rewrite mail addresses in canonical forms (see the `/etc/sendmail.cf` file in the configuration of the mail system) and the transition rules describing the behaviors of a tree automata. Rewriting is used in semantics in order to describe the meaning of programming languages as well as in program transformations like, for example, re-engineering of Cobol programs [vdBvDK+96]. It is used in order to compute [Der85], implicitly or explicitly like in Mathematica [Wol99] or OBJ [GKK+87], but also to perform deduction when describing by inference rules a logic [GLT89], a theorem prover [JK86] or a constraint solver [JK91]. It is of course central in systems making the notion of rule an explicit and first class object, like expert systems, programming languages based on equational logic, algebraic specifications, functional programming and transition systems.

In this very general picture, we introduce a calculus whose main design concept is to make all the basic ingredients of rewriting explicit objects, in particular the notions of rule *application* and *multisets of results*. We concentrate on *term* rewriting,

we introduce a very general notion of rewrite rule and we make the rule application and result explicit concepts. These are the basic ingredients of the *rewriting-* or *ρ-calculus* whose originality comes from the fact that terms, rules, rule application and therefore rule application strategies are all treated at the object level.

In ρ-calculus we can explicitly represent the application of a rewrite rule (say $a \to b$) to a term (like the constant a) as the object $[a \to b](a)$ which evaluates to the singleton $\{b\}$. This means that the rule application symbol $@$ (where $@$ is our notation for the placeholder) is part of the calculus syntax.

But the application of a rewrite rule may fail like in $[a \to b](c)$ that evaluates to the empty set $\emptyset$ or it can be reduced to a multiset with more than one element like exemplified later in this section and explained in Section 2.3. Of course, variables may be used in rewrite rules like in $[f(x) \to x](f(a))$. In this last case the evaluation mechanism of the calculus will reduce the application to $\{a\}$. In fact, when evaluating this expression, the variable x is bound to a via a mechanism classically called matching, and we recover the classical way term rewriting is acting.

Where this game becomes even more interesting is that $@ \to @$, the rewrite arrow operator, is also part of the calculus syntax. This is a powerful abstractor whose relationship with λ-abstraction [Chu40] could provide a useful intuition: A λ-expression $\lambda x.t$ could be represented in the ρ-calculus as the rewrite rule $x \to t$. Indeed the β-redex $(\lambda x.t\ u)$ is nothing else than $[x \to t](u)$ (i.e. the application of the rewrite rule $x \to t$ on the term u) which reduces to $\{\{x/u\}t\}$ (i.e. the application of the substitution $\{x/u\}$ to the term t).

So, basic ρ-calculus objects are built from a signature, a set of variables, the abstraction operator $@ \to @$, the application operator $@$, and we consider multisets of such objects. That gives to the ρ-calculus the ability to handle non-determinism in the sense of multisets of results. This is achieved via the explicit handling of reduction result multisets, including the empty set that records the fundamental information of rule application failure. 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 (match) this rewrite rule modulo commutativity the result is a set that contains two different elements corresponding to two possibilities.

To summarize, in ρ-calculus abstraction is handled via the arrow binary operator, matching is used as the parameter passing mechanism, substitution takes care of variable bindings and results multisets are handled explicitly.

The operational semantics of ELAN, a language based on labeled conditional rewrite rules and strategies controlling the rule application, can be described using the ρ-calculus. We use the ELAN language in order to describe and analyze the Needham-Schroeder public-key protocol [NS78].The implementation in ELAN is very concise and the rewrite rules describing the protocol are directly obtained from a classical presentation like the one given in Section 3.2.1.

2 Description of the ρ_T-calculus

We assume given in this section a theory T defined equationally or by any other means and we present the components of the ρ_T-calculus and we comment our main choices.

2.1 Syntax of the ρ_T-calculus

The *syntax* makes precise the formation of the objects manipulated by the calculus as well as the formation of substitutions that are used by the evaluation mechanism. In the case of ρ_T-calculus, the core of the object formation relies on a first-order signature together with rewrite rules formation, rule application and multisets of results.

Definition 2.1 We consider $\mathcal{X}$ a set of variables and $\mathcal{F} = \bigcup_m \mathcal{F}_m$ a set of ranked function symbols, where for all m, $\mathcal{F}_m$ is the subset of function symbols of arity m. We assume that each symbol has a unique arity i.e. that the $\mathcal{F}_m$ are disjoint. 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 can thus be inductively defined by the following grammar:

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

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

We adopt a very general discipline for the rewrite rule formation, and we do not enforce any of the standard restrictions often used in the rewriting community like non-variable left-hand-sides or occurrence of the right-hand-side variables in the left-hand-side. We also allow rewrite rules containing rewrite rules as well as rewrite rule application. We consider that the symbols $\{\}$ and $\emptyset$ both represent the empty set. For the terms of the form $\{t_1, \ldots, t_n\}$ we assume as usual that the comma is associative and commutative.

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 structure. 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 let us mention that the λ-terms and standard first-order rewrite rules [DJ90, BN98] are clearly objects of this calculus. For example, the λ-term $\lambda x.(y\ x)$ corresponds to the ρ-term $x \to [y](x)$ and a rewrite rule in first-order rewriting corresponds to the same rewrite rule in the rewriting-calculus.

We have chosen multisets as the data structure for handling the potential non-determinism. A multiset of terms could be seen as the set of distinct results obtained by applying a rewrite rule to a term. Other choices could be made depending on the intended use of the calculus. For example, if we do not want to provide the identical results of an application a set could be used. When the order of the computation of the results is important, lists could be employed. The confluence properties are

similar in a the set and multiset approaches. It is clear that for the list approach only a confluence modulo permutation of lists can be obtained.

Example 2.1 If we consider $\mathcal{F}_0 = \{a, b, c\}$, $\mathcal{F}_1 = \{f, g\}$, $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1$ and x, y variables in $\mathcal{X}$, some ρ-terms from $\varrho(\mathcal{F}, \mathcal{X})$ are:

- $[a \to b](a)$; this denotes the application of the rewrite rule $a \to b$ to the term a. We will see that the evaluation of this application is $\{b\}$.
- $[f(x, y) \to g(x, y)](f(a, b))$; a classical rewrite rule application leading to a $\{g(a, b)\}$result.
- $[y \to [x \to x + y](b)]([x \to x](a))$; a ρ-term that corresponds to the λ-term $(\lambda y.((\lambda x.x + y)\ b))\ ((\lambda x.x)\ a)$.
- $[[(x \to x + 1) \to (1 \to x)](a \to a + 1)](1)$; a more complicated ρ-term without corresponding standard rewrite rule or λ-term.

These examples show the very expressive syntax that is allowed for ρ-terms.

2.2 Matching and substitution application

The *matching algorithm* is used to bind variables to their actual values. In the case of ρ_T-calculus, this is in general higher-order matching. But in practical cases it will be higher-order-pattern matching, or equational matching, or simply syntactic matching and their combination. The matching theory is specified as a parameter (the theory T) of the calculus and when it is clear from the context this parameter is omitted.

Definition 2.2 For a given theory T over ρ-terms, 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 a 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. 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.

Notice that when the matching algorithm fails (i.e. returns $\mathbf{F}$) the function *Solution* returns the empty set.

Example 2.2 If $\ll^?_\emptyset$ denotes a syntactic matching and $\ll^?_C$ a commutative matching then we have:

1. $a \ll^?_\emptyset b$ has no solutions, and thus $\mathcal{S}olution(a \ll^?_\emptyset b) = \emptyset$;
2. $f(x, x) \ll^?_\emptyset f(a, b)$ has no solution and thus $\mathcal{S}olution(f(x, x) \ll^?_\emptyset f(a, b)) = \emptyset$;

3. $a \ll^?_\emptyset a$ is solved by all substitutions, and thus $\mathcal{S}olution(a \ll^?_\emptyset a) = \{\mathbb{ID}\}$;

4. $f(x, g(x, y)) \ll^?_\emptyset f(a, g(a, b))$ has as solution the substitution $\sigma \equiv \{x/a, y/b\}$, and $\mathcal{S}olution(f(x, g(x, y)) \ll^?_\emptyset f(a, g(a, b))) = \{\sigma\}$;

5. $x + y \ll^?_C a + b$ has the two solutions $\{x/a, y/b\}$ and $\{x/b, y/a\}$ and thus $\mathcal{S}olution(x + y \ll^?_C a + b) = \{\{x/a, y/b\}, \{x/b, y/a\}\}$.

The description of the *substitution application* on terms is often given at the meta-level, except for explicit substitution frameworks.

As for any calculus involving binders like the 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 ρ-calculus. In order to obtain a substitution that takes care of variable bindings we consider the usual notions of α-conversion and higher-order substitution as defined for example in [DHK00].

The burden of variable handling could be avoided by using an explicit substitution mechanism in the spirit of [CHL96]. We sketched such an approach in [CK99a] and this will be detailed in a forthcoming paper.

2.3 Evaluation rules of the ρ_T-calculus

The *evaluation rules* describe 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 evaluation rules of the ρ_T-calculus describe the application of a ρ-term on another one and specify the behavior of the different operators of the calculus when some arguments are multisets. They are defined in Figure 1.

In the rule $Fire$, $\{\sigma_1, \ldots, \sigma_i, \ldots\}$ represents the set of substitutions obtained by T-matching l on p (i.e. $\mathcal{S}olution(l \ll^?_T p)$) and $\sigma_i r$ represents the result of the application of the substitution σ_i on the term r. When the matching yields a failure represented by an empty set of substitutions, the result of the application of the rule $Fire$ is the empty set.

We should point out that, like in λ-calculus an application can always be evaluated, but unlike in λ-calculus, the set of results could be empty. More generally, when matching modulo a theory T, the set of resulting matches may be empty, a singleton (like in the empty theory), a finite set (like for associativity-commutativity) or infinite (like for associativity). We have thus chosen to represent the result of a rewrite rule application to a term as a multiset. An empty set means that the rewrite rule $l \to r$ fails to apply on t in the sense of a matching failure between l and t.

In order to push rewrite rule application deeper into terms, we introduce the two $Congruence$ evaluation rules. They deal with the application of a term of the form $f(u_1, \ldots, u_n)$ (where $f \in \mathcal{F}_n$) to another term of a similar 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 argument-wise. If the head symbols are not the same, an empty set is obtained.

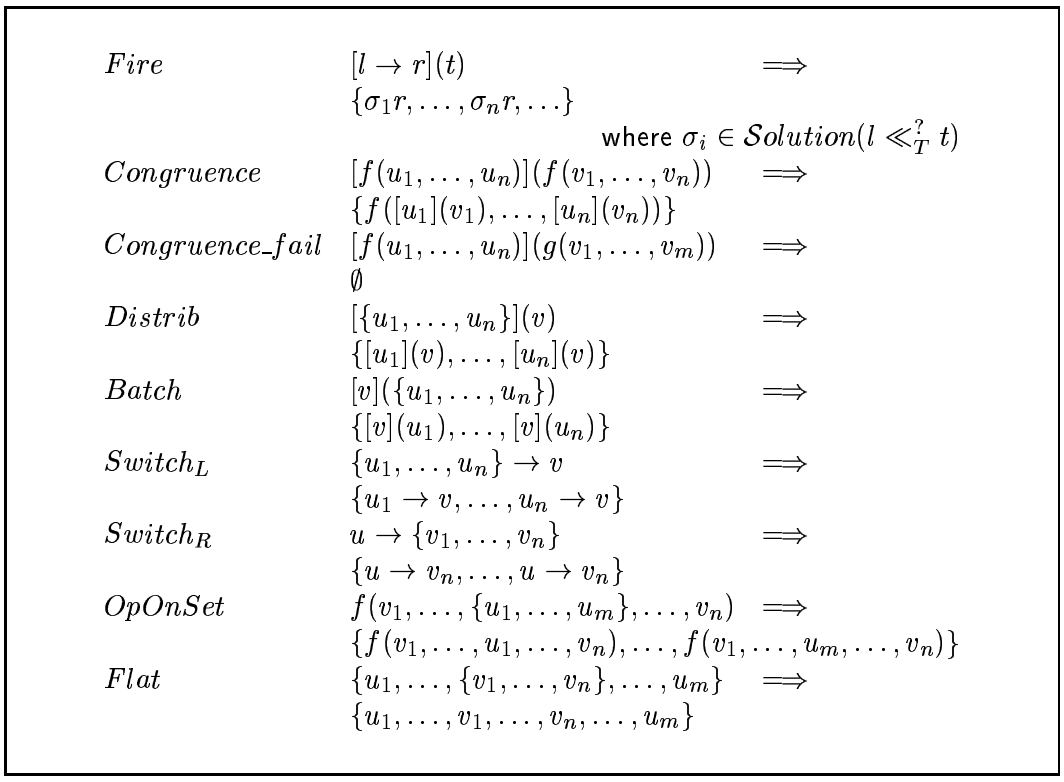

Figure 1: The evaluation rules of the ρ_T-calculus

The reductions corresponding to the cases where some sub-terms are multisets are defined by the last evaluation rules in Figure 1. These rules describe the propagation of the multisets on the constructors of the ρ-terms: the rules $Distrib$ and $Batch$ for the application, $Switch_L$ and $Switch_R$ for the abstraction and $OpOnSet$ for functions. The evaluation rule that corresponds to the multiset propagation for set symbols and that eliminates the redundant set symbols is the evaluation rule $Flat$.

This design decision to use multisets to represent reduction results has another important consequence concerning the handling of sets with respect to matching. Indeed, sets are just used to store results and we do not wish to make them part of the theory. We are thus assuming that the matching operation used in the $Fire$ evaluation rule is *not* performed modulo set axioms. This requires in some cases to use a strategy that pushes set braces outside the terms whenever possible.

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

2.4 Evaluation strategies for the ρ_T-calculus

The *strategy* guides the application of the evaluation rules. The strategy $\mathcal{S}$ guiding the application of the evaluation rules of the ρ_T-calculus could be crucial for obtaining good properties for the calculus. In a first stage, the main property analyzed is the confluence of the calculus and if the rule $Fire$ is applied under no conditions at any position of a ρ-term confluence does not hold.

The use of multisets for representing the reductions results is the main source of non-confluence. Unlike in the standard definition of a rewrite step where the rule application yields always a result, in ρ-calculus a rule application always yields a unique result that can be a multiset with several elements, representing the non-deterministic choice of the corresponding results from rewriting, or with no elements ($\emptyset$), representing the failure. Therefore, the relation generated by the evaluation rules of the ρ-calculus is finer and consequently non-confluent.

The confluence can be recovered if the evaluation rules of ρ-calculus are guided by an appropriate strategy. This strategy should first handle properly the problems related to the propagation of failure over the operators of the calculus. It should also take care of the correct handling of multisets with more than one element in non-linear contexts and details on this strategy are given in [CK99b].

2.5 Using the ρ_T-calculus

The aim of this section is to make concrete the concepts we have just introduced by giving a few examples of ρ-terms and ρ-reductions. Many other examples could be found on the ELAN web page [Pro00].

Let us start with the functional part of the calculus and give the ρ-terms representing some λ-terms. For example, the λ-abstraction $\lambda x.(y\ x)$, where y is a variable, is represented as the ρ-rule $x \to [y](x)$. The application of the above term to a constant a, $(\lambda x.(y\ x)\ a)$ is represented in the $\rho_\emptyset$-calculus by the application $[x \to [y](x)](a)$. This application reduces in the λ-calculus to the term $(y\ a)$ while in the $\rho_\emptyset$-calculus the result of the reduction is the singleton $\{[y](a)\}$. When a functional representation $f(x)$ is chosen, the λ-term $\lambda x.f(x)$ is represented by the ρ-term $x \to f(x)$ and a similar result is obtained. One should notice that for ρ-terms of this form (i.e. that have a variable as a left-hand side) the syntactic matching performed in the $\rho_\emptyset$-calculus is trivial, it never fails and gives only one result.

There is no difficulty to represent more elaborated λ-terms in the $\rho_\emptyset$-calculus. Let us consider the term $\lambda x.f(x)\ (\lambda y.y\ a)$ with the β-derivation: $\lambda x.f(x)\ (\lambda y.y\ a) \longrightarrow_\beta \lambda x.f(x)\ a \longrightarrow_\beta f(a)$. The same derivation can be recovered in the $\rho_\emptyset$-calculus for the corresponding ρ-term: $[x \to f(x)]([y \to y](a)) \longrightarrow_{Fire} [x \to f(x)](\{a\}) \longrightarrow_{Batch} \{[x \to f(x)](a)\} \longrightarrow_{Fire} \{\{f(a)\}\} \longrightarrow_{Flat} \{f(a)\}$. Of course, several reduction strategies can be used in the λ-calculus and reproduced accordingly in the $\rho_\emptyset$-calculus.

Now, if we introduce contextual information in the left-hand sides of the rewrite rules we obtain classical rewrite rules like $f(a) \to f(b)$ or $f(x) \to g(x)$. When we apply such a rewrite rule the matching can fail and consequently the application of the rewrite rule can fail. As we have already insisted in the previous sections, the

failure of a rewrite rule is not a meta-property in the $\rho_\emptyset$-calculus but is represented by an empty set (of results). For example, in standard term rewriting we say that the application of the rule $f(a) \to f(b)$ to the term $f(c)$ fails while in the $\rho_\emptyset$-calculus the term $[f(a) \to f(b)](f(c))$ evaluates to $\emptyset$.

When the matching is done modulo an equational theory we obtain interesting behaviors. Take, for example, the list operator $\circ$ that appends two lists with elements of sort *Elem*. Any object of sort *Elem* represents a list consisting of this only object.

If we define the operator $\circ$ as right-associative, the rewrite rule taking the first part of a list can be written in the associative ρ_A-calculus $l \circ l' \to l$ and when applied to the list $a \circ b \circ c \circ d$ gives as result the ρ-term $\{a, a \circ b, a \circ b \circ c\}$. If the operator $\circ$ had not been defined as associative we would have obtained as result of the same rule application one of the singletons $\{a\}$ or $\{a \circ b\}$ or $\{a \circ (b \circ c)\}$ or $\{(a \circ b) \circ c\}$, depending of the way the term $a \circ b \circ c \circ d$ is parenthesized.

Let consider now a commutative operator $\oplus$ and the rewrite rule $x \oplus y \to x$ that selects one of the elements of the tuple $x \oplus y$. In the commutative ρ_C-calculus the application $[x \oplus y \to x](a \oplus b)$ evaluates to the set $\{a, b\}$ that represents the set of non-deterministic choices between the two results. The rewrite rule $x \oplus y \to x$ applies as well on the term $a \oplus a$ and the result is the multiset $\{a, a\}$ representing the non-deterministic choice between the two elements that in this case represents two possible reductions with the same result. In a set approach the result of this latter reduction is $\{a\}$.

We can also use an associative-commutative theory like, for example, when an operator describes multiset formation. Let us go back to the $\circ$ operator but this time let us define it as associative-commutative and use the rewrite rule $x \circ x \circ L \to L$ that eliminates doubletons from lists of sort *Elem*. Since the matching is done modulo associativity-commutativity this rule eliminates the doubletons no matter what is their position in the multiset. For instance, in the ρ_{AC}-calculus the application $[x \circ x \circ L \to L](a \circ b \circ c \circ a \circ d)$ evaluates to $\{b \circ c \circ d\}$: the search for the two equal elements is done thanks to associativity and commutativity.

Another facility is due to the use of multisets for handling non-determinism. This allows us to easily express the non-deterministic application of a multiset of rewrite rules on a term. Let us consider, for example, the operator $\otimes$ as a syntactic operator. If we want the same behavior as before for the selection of each element of the couple $x \otimes y$, two rewrite rules should be non-deterministically applied like in the reduction: $[\{x \otimes y \to x, x \otimes y \to y\}](a \otimes b) \longrightarrow_{Distrib} \{[x \otimes y \to x](a \otimes b), [x \otimes y \to y](a \otimes b)\}$ $\longrightarrow_{Fire} \{\{a\}, \{b\}\} \longrightarrow_{Flat} \{a, b\}$.

As we have seen, the ρ-calculus can be used for representing some simpler calculi like λ-calculus and rewriting. This can be proved formally by restricting the syntax and the evaluation rules of the ρ-calculus in order to represent the terms of the two calculi. Thus, for any reduction in the λ-calculus or conditional rewriting a corresponding natural reduction in the ρ-calculus can be found. We can extend the encoding of conditional rewriting in the ρ-calculus to more complicated rules like the conditional rewrite rules with local assignments from the ELAN language.

3 Specifications in the ELAN language

3.1 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 [BKK+96]. It has been experimented on several non-trivial applications ranging from decision procedures, constraint solvers, logic programming and automated theorem proving but also specification and exhaustive verification of authentication protocols [Pro00]. ELAN's rewrite rules are conditional rewrite rules with local assignments. The local assignments are let-like constructions that allow applications of strategies on some terms. The general syntax of an ELAN rule is:

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

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

The application of the labeled rewrite rules is controlled by user-defined strategies while the unlabeled rules are applied according to a default normalization strategy. The normalization strategy consists in applying unlabeled rules at any position of a term until the normal form is reached, this strategy being applied after each reduction produced by a labeled rewrite rule.

The application of a rewrite rule in ELAN can yield several results due to the equational (associative-commutative) matching and to the `where` clauses that can return as well several results.

Example 3.1 An example of an ELAN rule describing a possible naive way to search the minimal element of a list by sorting the list and taking the first element is the following:

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

The strategy `sort` can be any sorting strategy. The operator `head` is supposed to be described by a confluent and terminating set of unlabeled rewrite rules.

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

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

`first`$(s_1, \ldots, s_n)$) that returns the results of the first unfailing strategy from its arguments.

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

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

Example 3.2 If the strategy `dk(x => x+1,x => x+2)` is applied on the term a, ELAN provides two results: $a+1$ and $a+2$. When the strategy `first(x => x+1,x => x+2)` is applied on the same term only the $a+1$ result is obtained. The strategy `first(b => b+1,a => a+2)` applied to the term a yields the result $a+2$.

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

A partial semantics could be given to an ELAN program using the rewriting logic [Mes92], 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.

3.2 Representing multisets in ELAN

Using non-deterministic strategies we can explore exhaustively the set of states of a given problem and find paths described by some specific properties. For example, for proving the correctness of the Needham-Schroeder authentication protocol [NS78] we look for possible attacks among all the behaviors during a session.

In the this section we briefly present some of the rules of the protocol and we give the strategy looking for all the possible attacks, a more detailed description of the implementation is given in [Cir99].

3.2.1 The Needham-Schroeder public-key protocol

The Needham-Schroeder public-key protocol [NS78] aims to establish a mutual authentication between an initiator and a responder that communicate via an insecure network. Each agent A possesses a *public key* denoted $K(A)$ that can be obtained by any other agent from a key server and a *(private) secret key* that is the inverse of $K(A)$. A message m encrypted with the public key of the agent A is denoted by $\{m\}_{K(A)}$ and can be decrypted only by the owner of the corresponding secret key, i.e. by A.

The protocol uses *nonces* that are fresh random numbers to be used in a single run of the protocol. We denote the nonce generated by the agent A by N_A.

The simplified description of the protocol presented in [Low95] is:

1. $A \rightarrow B$: $\{N_A, A\}_{K(B)}$
2. $B \rightarrow A$: $\{N_A, N_B\}_{K(A)}$
3. $A \rightarrow B$: $\{N_B\}_{K(B)}$

The initiator A seeks to establish a session with the agent B. For this A sends a message to B containing a newly generated nonce N_A and its identity, message encrypted with its key $K(B)$. When such a message is received by the agent B, he can decrypt it and extract the nonce N_A and the identity of the sender. The agent B generates a new nonce N_B and he sends it to A together with N_A in a message encrypted with the public key of A. When A receives this response he can decrypt it and assumes that he has established a session with B. The agent A sends the nonce N_B back to B and when receiving this last message B assumes that he has established a session with A since only A could have decrypted the message containing N_B.

The main property expected for an authentication protocol like the Needham-Schroeder public-key protocol is to prevent an intruder from impersonating one of the two agents.

The intruder is an user of the communication network and so, he can initiate standard sessions with the other agents and he can respond to messages sent by the other agents. The intruder can intercept any message from the network and can decrypt the messages encrypted with its key. The nonces obtained from the decrypted messages can be used by the intruder for generating new (fake) messages. The intercepted messages that can not be decrypted by the intruder can be replayed as they are.

3.2.2 Encoding the Needham-Schroeder public-key protocol in ELAN

We present now a description of the protocol in ELAN. The ELAN rewrite rules correspond to transitions of agents from one state to another after sending and/or receiving messages.

Data structures The initiators and the responders are agents described by their identity, their state and a nonce they have created. An agent can be defined in ELAN using a mixfix operator:

```
@ + @ + @  : ( AgentId SWC Nonce ) Agent;
```

The symbol `@` is a placeholder for terms of types `AgentId`, `SWC` and `Nonce` respectively representing the identity, the state and the current nonce of a given agent.

There are three possible values of `SWC` states. An agent is in the state `SLEEP` if he has not sent nor received a request for a new session. In the state `WAIT` the agent has already sent or received a request and when reaching the state `COMMIT` the agent has established a session.

A nonce created by an agent A in order to communicate with an agent B is represented by `N(A,B)`. Memorizing the nonce allows the agent to know at each moment who is the agent with whom he is establishing a session and the two identities

from the nonce are used when verifying the invariants of the protocol. A dummy nonce is represented by `N(di,di)`.

The nonces generated in the ELAN implementation are not random numbers but store some information indicating the agents using the nonce. If the uniqueness of nonces is important like, for example, in an implementation describing sequential runs of the protocol, an additional (random number) information can be easily added to the structure of nonces.

The agents exchange messages defined by:

```
@-->@:@[@,@,@] : (AgentId AgentId Key Nonce Nonce Address) message;
```

A message of the form `A-->B:K[N1,N2,Add]` is a message sent from `A` to `B` and contains the two nonces `N1` and `N2` together with the explicit address of the sender, `Add`. The address contains in fact the identity of the sender but we give it a different type in order to have a clear distinction between the identity of the sender in the encrypted part of the message and in the header of the message. The header of the message contains the source and destination address of the message but since they are not encrypted they can be faked by the intruder. The body of the message is encrypted with the key `K` and can be decrypted only by the owner of the private key.

The communication network is described by a possibly empty multiset of messages:

```
@       : ( message ) network;
@ & @   : ( network  network ) network (AC);
nill    : network;
```

with `nill` representing the network with no messages.

The intruder does not only participate to normal communications but can as well intercept and create (fake) messages. Therefore a new data structure is used for intruders:

```
@ # @ # @ : ( AgentId setNonce network ) intruder;
```

where the first field represents the identity of the intruder, the second one is the set of nonces he knows and the third one the set of messages he has intercepted. In our specification we only use one intruder and thus, the first field can be replaced by a constant identifying the intruder.

As for the messages, a set of nonces (`setNonce`) is defined using the associative-commutative operator `|` and a set of agents is defined using the associative-commutative operator `||`.

The ELAN rewrite rules are used to describe the modifications of the global state that consists of the states of all the agents involved in the communication and the state of the network. The global state is defined by:

```
@ <> @ <> @ <> @  :  ( setAgent setAgent intruder network ) state;
```

where the first two fields represent the set of initiators and responders, the third one represents the intruder and the last one the network.

Rewrite rules The rewrite rules describe the behavior of the honest agents involved in a session and the behavior of the intruder that tries to impersonate one of the agents. We will see that the invariants of the protocol are expressed by rewrite rules as well.

Each modification of the state of one of the participants to a session is described by a rewrite rule. At the beginning all the agents are in the state `SLEEP` waiting either to initiate a session or to receive a request for a new session.

When an initiator is in the state `SLEEP`, he initiates a session with one of the responders by sending the appropriate message as defined by the first step of the protocol. The following rewrite rule is used:

```
[initiator-1]
x+SLEEP+resp  || IN <> RE <> I <> lm                                  =>
x+WAIT+N(x,y) || IN <> RE <> I <> x-->y:K(y)[N(x,y),N(di,di),A(x)]&lm
    where (Agent)y+std+init :=(extAgent) elemIA(RE)                    end
```

In the above rewrite rule `x` and `y` are variables of type `AgentId` representing the identity of the initiator and the identity of the responder respectively. The initiator sends a nonce `N(x,y)` and his address (identity) encrypted with the public key of the responder and goes in the state `WAIT` where he waits for a response. Since only one nonce is necessary in this message, a dummy nonce `N(di,di)` is used in the second field of the message. The message is sent by including it in the multiset of messages available on the network.

Since the operator `||` is associative-commutative, when applying the rewrite rule `initiator-1` the initiator `x` is selected non-deterministicly from the set of initiators. The identity of the responder `y` is selected non-deterministicly from the set of responders or from the set of intruders; in our case only one intruder. The non-deterministic selection of the responder is implemented by the strategy `extAgent` that selects at each application a new agent from the set given as argument.

If the destination of the previously sent message is a responder in the state `SLEEP`, then this agent gets the message and decrypts it if it is encrypted with his key. Afterwards, he sends the second message from the protocol to the initiator and goes in the state `WAIT` where he waits for the final acknowledgement:

```
[responder-1]
IN<> y+SLEEP+init  || RE <>I<> w-->y:K(y)[N(n1,n3),N(n2,n4),A(z)]&lm
=> IN<> y+WAIT+N(y,z) || RE <>I<> y-->z:K(z)[N(n1,n3),N(y,z),A(y)]&lm
```

One should notice that due to the associative-commutative definition of the operator `&` the position of the message in the network is not important. A non-associative-commutative definition would have implied several rewrite rules for describing the same behavior.

The condition that the message is encrypted with the public key of the responder is implicitly tested due to the matching that instantiates the variable `y` from `y+SLEEP+init` and `K(y)` with the same agent identity. Therefore, we do not have to add an explicit condition to the rewrite rule that remains simple and efficient.

Two other rewrite rules describe the other message exchanges from a session. When an initiator `x` and a responder `y` have reached the state `COMMIT` at the end of

a correct session the nonce `N(y,x)` can be used as a symmetric encryption key for further communications between the two agents.

The intruder can be viewed as a normal agent that can not only participate to normal sessions but that tries also to break the security of the protocol by obtaining information that are supposed to be confidential. The network that serves as communication support is common to all the agents and therefore all the messages can be observed or intercepted and new messages can be inserted in it. There is no difficulty to implement the rules for the intruder in ELAN but for reasons of space they are omitted in this presentation.

The invariants of the protocol are easily represented by two rewrite rules describing the negation of the conditions that should be verified by the participants to the protocol session. If one of these two rewrite rules can be applied during the execution of the specification then the authenticity of the protocol is not ensured and an attack can be described from the trace of the execution.

Some additional properties on the multisets (of messages) can be expressed using unlabeled rewrite rules. For example the elimination of duplicates from a multiset of messages is represented by the rule

```
[] m & m & l => m & l
```

that is applied implicitly after each application of any labeled rule.

Strategies The rewrite rules used to specify the behavior of the protocol and the invariants should be guided by a strategy describing their application. Basically, we want to apply repeatedly all the above rewrite rules in any order and in all the possible ways until one of the attack rules can be applied.

The strategy is easy to define in ELAN by using the non-deterministic choice operator `dk`, the `repeat*` operator representing the repeated application of a strategy and the `;` operator representing the sequential application of two strategies:

```
[]attStrat  =>
  repeat*( dk(  attack-1, attack-2,
                intruder-1, intruder-2, intruder-3, intruder-4,
                initiator-1, initiator-2, responder-1, responder-2
  )); attackFound
```

The strategy tries to apply one of the rewrite rules given as argument to the `dk` operator starting with the rules for attacks and intruders and ending with the rules for the honest agents. If the application succeeds the state is modified accordingly and the `repeat*` strategy tries to apply a new rewrite rule on the result of the rewriting. When none of the rules is applicable, the `repeat*` operator returns the result of the last successful application. Since the `repeat*` strategy is sequentially composed with the `attackFound` strategy, this latter strategy is applied on the result of the `repeat*` strategy.

The strategy `attackFound` is nothing else but the rewrite rule:

```
  [attackFound]   ATTACK   =>   ATTACK              end
```

If an attack has not been found and therefore the strategy `attackFound` cannot be applied a backtrack is performed to the last rule applied successfully and another application of the respective rule is tried. If this is not possible the next rewrite rule is tried and if none of the rules can be applied a backtrack is performed to the previous successful application.

If the result of the strategy `repeat*` reveals an attack, then the `attackFound` strategy can be applied and the overall strategy succeeds. The trace of the attack can be recovered in the ELAN environment.

The trace obtained when executing the ELAN specification describes exactly the attack presented in [Low95] where the intruder impersonates an agent in order to establish a session with another agent.

The ELAN specification can be easily modified in order to reflect the correction shown sound in [Low96] and as expected, when the specification is executed with the modified rules no attacks are detected.

4 Conclusion

We have presented the ρ_T-calculus and we have seen that by making explicit the notion of rule, rule application and application result, the ρ_T-calculus allows us to describe in a simple yet very powerful manner the combination of algebraic and higher-order frameworks.

In the ρ_T-calculus the non-determinism is handled by using multisets of results and the rule application failure is represented by the empty set. Handling multisets is a delicate problem and the raw ρ_T-calculus, where the evaluation rules are not guided by a strategy, is not confluent but when an appropriate evaluation strategy is used the confluence is recovered.

The ρ_T-calculus is both conceptually simple as well as very expressive. This allows us to represent the terms and reductions from λ-calculus and conditional rewriting. Starting from this representation we showed how the ρ_T-calculus can be used to give a semantics to ELAN rules. This could be applied to many other frameworks, including rewrite based languages like ASF+SDF, ML, Maude or CafeOBJ but also production systems and non-deterministic transition systems.

We have shown how the ELAN language can be used as a logical framework for representing the Needham-Schroeder public-key protocol. This approach can be easily extended to other authentication protocols and an implementation of the TMN protocol has been already developed. The rules describing the protocol are naturally represented by conditional rewrite rules. The mixfix operators declared as associative-commutative allow us to express and handle easily the random selection of agents from a set of agents or of a message from a set of messages.

Among the topics of further research, let us mention the deepening of the relationship between the ρ_T-calculus and the rewriting logic [Mes92], the study of the models of the ρ_T-calculus, and also a better understanding of the relationship between the rewriting relation and the rewriting calculus.

References

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

[BN98] F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Cir99] H. Cirstea. Specifying authentication protocols using ELAN. In *Workshop on Modelling and Verification*, Besancon, France, December 1999.

[CK99a] H. Cirstea and C. Kirchner. Combining higher-order and first-order computation using ρ-calculus: Towards a semantics of ELAN. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, Research Studies, ISBN 0863802524, pages 95–120. Wiley, 1999.

[CK99b] H. Cirstea and C. Kirchner. An introduction to the rewriting calculus. Research Report RR-3818, INRIA, December 1999.

[Der85] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65(2/3):122–157, 1985.

[DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[DJ90] 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.

[GKK+87] J. A. Goguen, C. Kirchner, H. Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, July 1987. Also as internal report CRIN: 88-R-001.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[JK86] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City (USA), 1984.

[JK91] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In J.-L. 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.

[Low95] G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public key protocol using CSP and FDR. In *Proceedings of 2nd TACAS Conf.*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, Passau (Germany), 1996. Springer-Verlag.

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

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

[Pro00] Protheo Team. The ELAN home page. WWW Page, 2000. `http://www.loria.fr/ELAN`.

[vdBvDK+96] M. van den Brand, A. van Deursen, P. Klint, S. Klusener, and E. A. van der Meulen. Industrial applications of asf+sdf. In M. Wirsing and M. Nivat, editors, *AMAST '96*, volume 1101 of *Lecture Notes in Computer Science*, pages 9–18. Springer-Verlag, 1996.

[Wol99] S. Wolfram. *The Mathematica Book*, chapter Patterns, Transformation Rules and Definitions. Cambridge University Press, 1999. ISBN 0-521-64314-7.