REVEUR-3: THE IMPLEMENTATION OF A GENERAL COMPLETION PROCEDURE PARAMETERIZED BY BUILT-IN THEORIES AND STRATEGIES*

Claude KIRCHNER and Hélène KIRCHNER
Centre de Recherche en Informatique de Nancy, 54506 Vandœuvre les Nancy Cedex, France

Communicated by M.C. Gaudel
Received September 1985
Revised March 1986

Abstract. This paper describes REVEUR-3, a software that implements a general completion procedure. REVEUR-3 allows working with built-in theories and strategies and is aimed to perform proofs and experiments in term rewriting systems. These features are illustrated by experimental results.

1. Introduction

The completion procedure (see Buchberger [3], Knuth and Bendix [23]) originally computes a term rewriting system $R$ which generates the same equivalence on terms as a given set of equations $A$. Moreover $R$ is proved to have the so-called Church-Rosser property, which allows deciding equality of two terms by using rewritings only, provided $R$ terminates. A proof method for $A$-equality is derived which consists of computing irreducible forms of the two terms of an equational theorem and checking for their identity.

This method was extended to handle the case of an equational term rewriting system, that is a pair composed of a set of rules $R$ and a set of axioms $E$. A first approach due to Lankford and Ballantyne [26, 27, 25] handles the case of permutative axioms that generate finite $E$-congruence classes. The case of infinite $E$-congruence classes is studied in [11, 32, 15]. Huet's approach is restricted to sets $R$ of left-linear rules, while Peterson and Stickel's one is restricted to theories $E$ defined by left and right-linear axioms and for which a finite and complete unification algorithm is known. These results were unified by Jouannaud who described the underlying computations used in both approaches. Moreover his results allowed dropping all the previous linearity conditions. Based on these ideas, a very general completion procedure was described in [16] that subsumes all known untyped completion algorithms.

* This research was partly supported by the GRECO of programmation and by ADI under Grant 82/767. The paper was revised during the stay of the first author at SRI International and the second one at CSLI (Stanford University).
The purpose of this paper is to describe an implementation of this procedure, perform experiments and compare them. Our experimental version is hereafter called REVEUR-3 since it has been designed as an extension of the REVE software, written in CLU [28]. REVE is a rewrite rule laboratory, that is a generator of term rewriting systems, based on the completion procedure. Among many interesting features, it provides automatic or semi-automatic proofs of termination of the generated rewriting system. Two previous versions of REVE are REVE-1 [30] and REVE-2 [7] which is currently the distributed version of REVE. REVEUR-3 is a rather large program (about 25,000 lines of annotated code) running on VAX and SUN machines, under the UNIX system.

After a review of the theoretical framework in Section 2, we emphasize in Section 3 the originality of REVEUR-3 and describe some of our experiments in the last section.

2. Theoretical framework

This paper is aimed at readers who are familiar with the basic notions of term rewriting systems and the completion procedure, including terms, occurrences, substitutions, equational equality generated by a set of axioms $E$ (denoted $\sim_E$), rewriting rule, rewriting system, normal form of a term $t$ (hereafter denoted $t\downarrow$), critical pair. Definitions of these concepts can be gleaned from [12].

All the theoretical bases of this section can be found in [16] or [22]; we only remind here the main concepts.

2.1. Equational rewriting

REVEUR-3 allows equational rewriting i.e. uses mixed sets of rules and equations. This need comes from the fact that some permutative equations, such as commutativity, cannot be oriented into rewrite rules without compromising the termination of the rewriting process. In order to take into account such equations, the first idea is to work on equivalence classes of terms. Thus if $E$ is the set of (non-directed) axioms, the set of terms is quotiented by the equivalence relation $\sim_E$. A set of rewrite rules $R$ induces then a reduction relation on equivalence classes:

$$T \rightarrow T' \iff (\exists t \in T, \exists t' \in T', t \rightarrow_R t')$$

where $\rightarrow_R$ is the standard rewriting relation on terms defined by:

$$t \rightarrow_R t' \text{ iff there exist a subterm } t_1 \text{ of } t \text{ at occurrence } u,$$

a substitution $\sigma$,

and a rule $g \rightarrow d$ in $R$

---

1. The CLU licence for VAX and SUN and a distribution tape can be obtained from MIT (B. Liskov) or from CRIN (P. Lescanne).

2. The current REVE-2 distribution tape can be obtained from MIT (J. Guttag) or from CRIN (P. Lescanne).

3. The current version of REVEUR-3 can be asked to the authors.
such that \( t_1 = \sigma(g) \),
and \( t' = f_{u \rightarrow \sigma(d)} \) (which denotes the term \( t \)
where \( t_1 \) has been replaced by \( \sigma(d) \)).

The reduction relation on \( E \)-equivalence classes of terms cannot be efficiently
implemented except perhaps in some particular cases. Moreover it can be undecid-
able when \( E \)-equivalence classes are infinite. Thus different attempts have been
made in order to define a rewriting relation on terms that simulates the reduction
relation \( \rightarrow \).

Peterson and Stickel proposed a second type of term rewriting, called rewriting
modulo \( E \), that uses an \( E \)-matching algorithm:

\[
t \rightarrow_{R,E} t' \text{ iff there exist a subterm } t_1 \text{ of } t \text{ at occurrence } u,
\text{ a substitution } \sigma,
\text{ and a rule } g \rightarrow d \text{ in } R
\text{ such that } t_1 \sim\!_E \sigma(g)
\text{ and } t' = f_{u \rightarrow \sigma(d)}.\]

Another term rewriting relation has been proposed by Jouannaud to combine
these two ones. Splitting the set of rules into two parts, left-linear rules \( L \) and
non-left-linear ones \( N \), he defines \( \rightarrow_{L \cup N,E} \) as either a rewriting using a rule of \( L \) or
a rewriting modulo \( E \) using a rule of \( N \). This idea allowed him to generalize previous
results of Huet and of Peterson and Stickel.

For any binary relation \( \rightarrow \), let us denote by \( \leftrightarrow \) its reflexive transitive closure and by \( \tilde{\leftrightarrow} \) its reflexive symmetric transitive closure.

The reduction relation \( \rightarrow \) on \( E \)-equivalence classes of terms is said to be Church-
Rosser iff

\[
T \tilde{\leftrightarrow} T' \Rightarrow (\exists T'', T \tilde{\leftrightarrow} T'' \tilde{\leftrightarrow} T').
\]

In Huet, Peterson and Stickel and Jouannaud’s approaches, other Church-Rosser
properties are used. They are defined on terms and no more on \( E \)-equivalence
classes of terms. All of them imply the Church-Rosser property on \( E \)-equivalence
classes of terms and all of them assume the termination of \( \rightarrow \).

2.2. Correspondence between rewriting relation and the Church-Rosser property

The following table shows, for each previously mentioned approach, the corre-
spondence between the rewriting relation and the Church-Rosser property:

<table>
<thead>
<tr>
<th>Knuth and Bendix’s method:</th>
<th>( t \sim_{R \cup E} t' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E ) empty</td>
<td>( \rightarrow_R )</td>
</tr>
<tr>
<td>Huet’s method:</td>
<td>( t \sim_{R \cup E} t' )</td>
</tr>
<tr>
<td>( R ) left-linear rules</td>
<td>( \rightarrow_R )</td>
</tr>
</tbody>
</table>

iff \( \exists t'' \text{ s.t. } t \tilde{\rightarrow}_R t'' \tilde{\leftrightarrow} t' \).

iff \( \exists t_1'', t_2'' \text{ s.t. } t \tilde{\rightarrow}_R t_1'' \sim\!_E t_2'' \tilde{\leftrightarrow} t' \).
Peterson and Stickel's method:  
\[ t \sim_{R,E} t' \]
\[ \text{iff } \exists t_1^*, t_2^* \text{ s.t. } t \xrightarrow{R,E} t_1^* \xrightarrow{E} t_2^* \xrightarrow{R,E} t'. \]

Jouannaud's method:  
\[ t \sim_{R,E} t' \]
\[ \text{iff } \exists t_1^*, t_2^* \text{ s.t. } t \xrightarrow{L,N,E} t_1^* \xrightarrow{E} t_2^* \xrightarrow{L,N,E} t'. \]

Let us emphasize that these Church-Rosser properties are actually similar up to the rewriting relation used. If \( \rightarrow_{R,E} \) is any rewriting relation taking into account rules and equations, the corresponding Church-Rosser property is
\[ t \sim_{R \cup E} t' \iff (\exists t_1^*, t_2^* \text{ such that } t \xrightarrow{R,E} t_1^* \xrightarrow{E} t_2^* \xrightarrow{R,E} t'). \]

In the following, we denote this property as the "RE-Church-Rosser property".

As soon as it is satisfied and \( \rightarrow_{R,E} \) terminates, we get a decision procedure for \((R \cup E)\)-equality by computing the RE-normal forms of \( t \) and \( t' \) and testing their \( E \)-equality. Thus to each choice of a \( \rightarrow_{R,E} \) rewriting relation, corresponds a theorem providing method. But notice the increasing power of the rewriting relations:

\[ \rightarrow_L \subseteq \rightarrow_{L \cup N,E} \subseteq \rightarrow_{(L \cup N) \cup E}. \]

Thus we get corresponding sorted Church-Rosser properties and a classification of the different equational proof methods when there are axioms. For example, we can say that in general Huet's rewriting method is less powerful than Jouannaud's one with the following meaning: any equational theorem that can be proved with \( \rightarrow_L \) as rewriting method can also be proved with \( \rightarrow_{L \cup N,E} \). This last rewriting method is itself less powerful than Peterson and Stickel's one. Nevertheless let us already mention that the implementation of a rewrite rule laboratory providing the different possibilities allowed us to compare these methods with respect to other criteria, as described in the last section.

2.3. Theoretical problems

The theoretical study of the RE Church-Rosser property makes clear that two properties, namely confluence and coherence are necessary and sufficient conditions, assuming the termination of the reduction relation \( \rightarrow \) on \( E \)-equivalence classes and provided these classes are finite.

In addition to confluence, coherence is required to enable computations in \( E \)-equivalence classes; more precisely coherence is the necessary and sufficient condition for all the RE-normal-forms of any term \( t \) to belong to the same \( E \)-equivalence class.

Coherence and confluence can be checked on an adapted notion of critical pairs. Thus when working modulo a set of axioms \( E \), critical pairs must be computed both between rules (confluence critical pairs) and between rules and axioms (coherence
critical pairs). The first ones must satisfy the confluence property, the second ones the coherence property depicted below:

**Confluence.** For any confluence critical pair \((p, q)\), there exist \(p'\) and \(q'\) such that:

\[ p \xrightarrow{\text{RE}} p' \sim E q' \xrightarrow{\text{RE}} q. \]

**Coherence.** For any coherence critical pair \((p, q)\), there exist \(p'\), \(q'\), \(q_1\) such that:

\[ p \xrightarrow{\text{RE}} p' \sim E q' \xrightarrow{\text{RE}} q_1 \xrightarrow{\text{RE}} q. \]

Notice that coherence needs to reduce at least once the right-hand side of a coherence critical pair obtained for instance by overlapping a rule \(l \rightarrow r\) into an axiom \(g = d\) at occurrence \(u\). This term \(q\) is an instance of \(d\), denoted \(\sigma(d)\). The difficulty comes from the fact that the rule that is used to perform the reduction of \(q\) at some step of the completion process, can be later removed and replaced by a new one, during the simplification process used to inter-reduce rules. Consequently, to ensure coherence, it can be necessary to protect an existing rule which reduces a right-hand side of a coherence critical pair. When no such rule exists, it is necessary to introduce a new rule \(q \rightarrow p\) or an extension. The extension introduced for a non-left-linear rule \(l \rightarrow r\) is defined as the rule \(g_{u \rightarrow l} \rightarrow g_{u \rightarrow r}\). Notice that this definition generalizes the Peterson and Stickel’s extensions and that such a rule reduces the right-hand side of the corresponding coherence critical pair because \(\sigma(d) \sim_l \sigma(g) \sim_l \sigma(g_{u \rightarrow l})\). Except when the rule \(l \rightarrow r\) is deleted, neither extensions nor protected rules are allowed to be deleted from the rewriting system, since this would compromise the Church-Rosser property.

Up to now two methods have been proposed and implemented in order to ensure the coherence:

- The first one consists of testing reducibility of the right-hand side of coherence critical pairs with already existing rules. This method avoids introducing useless extensions but needs to protect some rules.
- The second method consists of automatically adding extensions for any rule in the system. The form and the number of extensions can be deduced from the axioms and the rule itself. In the associative-commutative theory case, it is proved that it is not necessary to introduce extensions of extensions. But in general, this method possibly leads to add infinitely many extensions.

Another theoretical problem arises when trying to work with theories \(E\) with infinite equivalence classes. Putting in \(E\) an axiom like idempotency \((x + x = x)\), or involution \((-x) = x\), leads to this situation. The tools developed in this case are yet more complex and up to now, we only got a sufficient condition to ensure the Church-Rosser property, which seems a little too strong [16].

Finally, let us point out that orientation of equations into rules is also more difficult in the equational case. More precisely, what is needed is the termination of the reduction relation \(\rightarrow\) on \(E\)-equivalence classes, called \(E\)-termination property.
Little is known about this property. A theoretical study of the problem can be found in [17] and effective, yet complex methods for associative-commutative theories in [4]. More recent results in this last case are given in [1] and [9].

2.4. A general completion procedure

The aim of a completion procedure is to compute from a set of equations \( P \) and a set of axioms \( E \), a set of rewrite rules \( R \) such that \( \sim_{R \cup E} \) is equal to \( \sim_{P \cup E} \) and such that a Church-Rosser property is satisfied.

We give here a tail recursive form of the general completion procedure implemented in REVEUR-3, see Fig. 1. It works with a set of rules \( R \), a set of equations \( P \) and a possibly empty set \( E \) of axioms. Axioms of \( E \) can be seen as defining a theory or as defining properties of operators. For example, \( E \) can define an associative-commutative theory, that is more precisely one or more operators \(*\), satisfying the following axioms:

\[
(x * y) = (y * x), \quad ((x * y) * z) = (x * (y * z)).
\]

Contrary to the fixed set \( E \), \( P \) is a set of equations which evolves during the completion process. It is the set of equations which have to be directed into rules. For that purpose, a well-founded reduction ordering \( > \) is provided, which must be compatible with the \( E \)-equivalence classes (i.e. \( t \sim_E t' \Rightarrow t < t' \) and \( t' < t \)).

The SIMPLIFICATION procedure modifies both \( P \) and \( R \). Whenever a new rule is introduced in \( R \), it is used to reduce other non-protected rules and equations. A rule whose left-hand side is reduced becomes a new equation in \( P \).

The CRITICAL-PAIRS procedure computes overlappings between a rule \( l \rightarrow r \) and other rules in \( R \) and between the rule and axioms of \( E \), creating new equations in \( P \) by this way. It also adds extensions and protects rules in \( R \) if needed.

Both procedure are described with more details in [16].

The COMPLETION procedure is said general because any known untyped completion procedure can be expressed as a particular instance of it, using parameterization at different levels. First the procedure can be parameterized by the set of axioms \( E \). Second, the four main operations in a completion procedure are:

- normalization of terms,
- orientation of equations into rules,
- simplification of other rules and equations using the new added rules,
- computation of critical pairs between rules and between rules and axioms.

For each of them, changing some parameters leads to a different behavior of the completion process. For example, the choice of the rewriting relation used in the normalization implies a specific Church-Rosser property and thus a new proof method for deciding equational equality. Thus parameterization can also be introduced at the level of these basic operations. We now develop this idea which appears as an originality of REVEUR-3.
PROCEDURE E-COMPLETION(P, R, E, >)
if P is not empty
then choose a pair (p, q) in P
   \[ p' = p \text{;} q' = q \]
   case \( p' \sim_E q' \) then E-COMPLETION(P - \{(p, q)\}, R, E, >)
      if \( p' > q' \) then (P, R) = SIMPLIFICATION(P - \{(p, q)\}, R, p' + q')
      E-COMPLETION(P, R \cup \{p' + q\}, E, >)
     else (P, R) = SIMPLIFICATION(P - \{(p, q)\}, R, q' + p')
      E-COMPLETION(P, R \cup \{q' + p\}, E, >)
   end case
else STOP with FAILURE
end if
endif

END E-COMPLETION

Fig. 1. The completion procedure.

3. Originality of REVEUR-3

Our main objective was to implement a general completion procedure that allows both built-in equational theories \( E \) and experimentation of the different completion processes mentioned before. This aim implies the modularity of the system in order to allow easy changes and enrichments.

Thus from the user’s external point of view the software provides different functionalities and seems to have different behaviors. In this way it is a rewrite rule laboratory in the same vein as its predecessors REVE-1 and REVE-2. On the contrary, from the designer’s internal point of view, it appears as a general and unified procedure. Let us describe more precisely these ideas.

3.1. Built-in equational theories

The parametrization of the general completion procedure by the set of axioms \( E \) involves generalizations of three basic operations: equality decision, matching and unification. All of them have to take into account equational properties of some function symbols. For example, a symbol \(+\) may be associative and commutative, another left-distributive on \(+\), a third one may have no property. To each property...
corresponds a set of axioms that is explicitly used in the completion procedure to compute coherence critical pairs. On the other hand, equality decision, matching and unification use the properties of operators and work on terms which are built from all these symbols.

In [20, 19], it is shown that such a unification procedure can be designed in a very general way. It is based on two main procedures. The first one decomposes the initial problem of unification into a set of unification problems all of the form $x = t$ where $x$ is a variable and $t$ a term. The second one solves the resulting unification problems. The first process uses the three following operations:

- Decomposition of equations determines their common part and their sets of disagreements: for example, if $f, g$ are symbols which have no property, the equation $x = f(z, g(y)) = f(g(a), z)$ is decomposed into the equivalent system $F = \{x = f(z, z), z = g(a), z = g(y)\}$.

- Merging all the conditions on the same variable, for example the previous system $F$ is merged into $\{x = f(z, z), z = g(a) = g(y)\}$, and decomposition can further be applied on the second equation.

- Mutation replaces a no longer decomposable equation, say $(t = t')$, by a set of systems of equations from which a complete set of solutions of $(t = t')$ can be easily deduced. Mutation works on equations $(t = t')$ where the top symbols of $t$ and $t'$ have the same equational property and thus is based on specific processes in the built-in equational theories. For example if the symbol $+$ is supposed to be commutative, then the equation $s + a = y + b$ is replaced by $\{(s = y, a = h), (s = h, a = y)\}$. A similar approach works for matching [29].

Thus working with built-in theories means to attach equational properties to function symbols, to introduce explicit axioms and to design specific processes used for equality decision, matching and unification. In REVEUR-3 a built-in equational theory has been implemented as a module composed of axioms and of special procedures: equality, matching, unification. The name of the theory is attached to the operators satisfying the axioms. Up to now, the empty theory ($E$ is the empty set of axioms), the commutative and the associative-commutative theories are implemented and can be mixed.

3.2. Strategies

A strategy is a set of parameters that determine a particular behavior of the completion process. We have grouped together under this concept several more or less original ideas.

- From our theoretical study of $E$-completion, the idea arises to design a system which can deal with different rewriting relations and use different methods to test the coherence property.

- The state of art about automatic orientation compelled us to introduce at least an interactive way to orient them. The introduction of a possible choice between
a manual and an automatic orientation has been comforted by some other experiments with REVE \cite{30,2}.

- In the same vein, we have been convinced that it can be useful to choose a particular superposition strategy between rules either for efficiency reasons or in order to perform experiments.

In REVEUR-3, different parameters of a completion process may be set, according to the user's choices, and according to them, the system has a different behavior. We review in this section three kinds of choices which are effectively implemented. Of course a lot of other ones could be added.

3.2.1. Choice of the rewriting relation

Let us explain more precisely how we have implemented the choice of the rewriting relation. The completion procedure always works with two sets of rules named $R_1$ and $R_2$. In general, standard rewriting is performed using rules of $R_1$, while rewriting modulo the axioms $E$ is performed using rules of $R_2$. Now, according to the different Church-Rosser properties the user may choose, the sets of rules are built as in Fig. 2.

The choice of the rewriting relation is thus entirely implemented only by the way of introducing rules into $R_1$ or $R_2$.

<table>
<thead>
<tr>
<th>Knuth and Bendix's method:</th>
<th>all rules are put in $R_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$ empty</td>
<td>R left-linear rules</td>
</tr>
<tr>
<td>Huet's method:</td>
<td>only left-linear rules are allowed and</td>
</tr>
<tr>
<td>$R$ left-linear rules</td>
<td>put in $R_1$</td>
</tr>
<tr>
<td>Peterson and Stickel's method:</td>
<td>all rules are put in $R_2$</td>
</tr>
<tr>
<td>$E$ linear axioms</td>
<td></td>
</tr>
<tr>
<td>Jouannaud's method:</td>
<td>left-linear rules are put in $R_1$</td>
</tr>
<tr>
<td>$L$ left-linear rules</td>
<td>non-left-linear rules are put in $R_2$</td>
</tr>
<tr>
<td>$N$ non left-linear rules</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Construction of the set of rules.

3.2.2. Choice of the coherence check

According to the rewriting relation chosen by the user, a strategy for ensuring the coherence may be proposed. The choice arises in the way to add extensions. Of course, with an empty set of axioms $E$ or Huet's method, there is no need to add extensions. But with other methods the user can choose between checking the coherence with already existing rules, or systematically adding extensions for the rules of $R_2$. As mentioned before, this last method, actually chosen by Peterson and Stickel, is valid with associative-commutative theories. Thus the user who wants to make experiments in these theories may choose between the two possibilities and
the completion process will use the corresponding procedure for coherence critical pairs.

3.2.3. Choice of the superposition strategy

Two superposition strategies are provided in order to overlap rules: each rule with all the previously introduced (or older) ones, or each rule with smaller ones. Since each rule is superposed with all rules which precede it in the list of rules, changing the superposition strategy is equivalent to change the way to classify rules when they are introduced in the list. If the list is sorted in decreasing order according to the age of the rule, the superpositions will be made with all the previously introduced rules. Whereas if the list is sorted by increasing order with respect of the size of the rules, each rule will be superposed with smaller ones.

3.3. More on implementation

Beyond the design of general procedures for equality decision, matching and unification working with built-in theories, some other generalizations are required for the general completion procedure we propose.

Problems are due to the complexity of the \( E \)-completion process: some procedures, especially normalization and simplification, need standard rewriting associated with a first subset of rules and rewrites modulo \( E \) associated with another subset. Thus the reduction process itself is parameterized by the type of matching, which can be either usual matching, or \( E \)-matching. The same feature appears at the critical pairs level, where unification must be performed sometimes in the empty theory, sometimes in the \( E \) theory.

On the other hand, let us mention that a complex implementation of a rewriting system was needed because checking the coherence property needs to add extensions and to protect some rules. Accordingly the simplification process of the rewriting system by the new introduced rules becomes much more complex and needs access to extensions and to protected rules.

4. Experiments

We present in this section some examples which allow comparisons between different strategies. From experiments the following idea arises: according to the strategies used in REVEUR-3, the same starting set of equations can be completed using different methods which are more or less powerful with respect to some criteria. A first criterion is the termination of the completion process: a strategy which allows finding a finite rewriting system \( R \) such that \((R \cup E)\) is equivalent to the starting equational theory can be considered as more interesting than a strategy with which the completion process fails with a non orientable equation or generates an infinite set of rewrite rules. A second criterion illustrated in our examples is the
time consumed by the completion process. This time is strongly related to the efficiency of the rewriting relation which can be considered as a third criterion; it is clear that rewriting modulo $E$ is very expensive and less it is used more efficient is the rewriting. Thus according to the efficiency criterion, we can say that $\rightarrow_{L \cup N, E}$ is more efficient than $\rightarrow_{(L \cup N), E}$.

We now study on four examples the effect of changing the rewriting strategy on the generated set of rules.

### 4.1. Abelian groups

For abelian groups, Lankford and Ballantyne together with Peterson and Stickel have found a term rewriting system satisfying the $R,E$-Church-Rosser property. The same experiment was performed with REVEUR-3. Our completion process was initialized with the rewriting relation $\rightarrow_{R,E}$ and the following sets of axioms and equations:

You are currently working modulo the following axioms:

\[
\begin{align*}
((x + y) + z) &= (x + (y + z)) \\
(x + y) &= (y + x)
\end{align*}
\]

User equations:

1. $(x + 0) = x$
2. $(x + i(x)) = 0$

No critical pair equations.

No rewrite rule in $R_1$.

No rewrite rule in $R_2$.

REVEUR-3 terminates with the message:

Your are currently working modulo the following axioms:

\[
\begin{align*}
((x + y) + z) &= (x + (y + z)) \\
(x + y) &= (y + x)
\end{align*}
\]

No rewrite rule in $R_1$.

Rewrite rules in $R_2$:

1. $i(0) \rightarrow 0$
2. $(x + 0) \rightarrow x$
   
   Which has for extensions:
3. $(z + (x + 0)) \rightarrow (z + x)$
4. $i(i(x)) \rightarrow x$
5. $(x + i(x)) \rightarrow 0$
Which has for extensions:
6. \((z + (x + i(x))) \rightarrow z\)
7. \(i((x + z)) \rightarrow (i(z) + i(x))\)

Your system is complete!

Using now the rewriting relation \(\rightarrow_{\text{L,N,E}}\), REVEUR-3 terminates with the message:

Your are currently working modulo the following axioms:

\((x + y) + z = (x + (y + z))\)
\((x + y) = (y + x)\)

Rewrite rules in R1:
1. \(i(0) \rightarrow 0\)
2. \((x + 0) \rightarrow x\)
3. \((0 + x) \rightarrow x\)
4. \(i(i(z)) \rightarrow z\)
5. \(i((z + x)) \rightarrow (i(z) + i(x))\)

Rewrite rules in R2:
6. \((x + i(x)) \rightarrow 0\)

Which has for extensions:
7. \(((x + i(x)) + z) \rightarrow z\)

Your system is complete!

Notice that now the rule \(0 + x \rightarrow x\) is needed to insure equivalence between \(\rightarrow\) and \(\rightarrow_{\text{L,N,E}}\)-reducibility. The second completion is twice faster than the first one. It is due to the fact that less associative-commutative unification and matching are used in the second case. This result is new in the following way: it provides a rewriting relation (here \(\rightarrow_{R1 \cup R2,E}\)) which allows deciding equality in abelian groups and which is more efficient than the previous one proposed by Peterson and Stickel.

4.2. Commutative monoid with two generators and identity

Let us now consider the set of equations:

\(0 + x = x\)
\(a + b = 0\)
\((x + a) + b = x\)

and assume the associativity and commutativity of the + symbol. Using \(\rightarrow_{\text{K,E}}\) as rewriting relation, REVEUR-3 terminates with the message:

Your are currently working modulo the following axioms:

\(((x + y) + z) = (x + (y + z))\)
\((x + y) = (y + x)\)

No rewrite rule in R1.
Rewrite rules in R2:

1. \((0 + x) \rightarrow x\)
   Which has for extensions:
2. \((z + (0 + x)) \rightarrow (z + x)\)

3. \((a + b) \rightarrow 0\)
   Which has for extensions:
4. \((z + (a + b)) \rightarrow z\)

5. \(((x + a) + b) \rightarrow x\)
   Which has for extensions:
6. \((z + ((x + a) + b)) \rightarrow (z + x)\)

Your system is complete!

But using \(-L\) or \(-L_{\cup N.E}\) as rewriting relation, we get an infinite set of rules whose first ones are described in the following message:

Your are currently working modulo the following axioms:

\[((x + y) + z) \equiv (x + (y + z))\]
\((x + y) \equiv (y + x)\)

Rewrite rules in R:

1. \((0 + x) \rightarrow x\)
2. \((x + 0) \rightarrow x\)
3. \((a + b) \rightarrow 0\)
4. \((b + a) \rightarrow 0\)
5. \(((x + a) + b) \rightarrow x\)
6. \((a + (b + z)) \rightarrow z\)
7. \((b + (a + z)) \rightarrow z\)
8. \(((x + b) + a) \rightarrow x\)
9. \((b + (x + a)) \rightarrow x\)
10. \(((a + x) + b) \rightarrow x\)
11. \(((b + z) + a) \rightarrow z\)
12. \((a + (y + b)) \rightarrow y\)
13. \(((x + a) + (b + z)) \rightarrow (x + z)\)
14. \(((x + (y + a)) + b) \rightarrow (x + y)\)
15. \((a + ((b + z) + z1)) \rightarrow (z + z1)\)

\[ \ldots \]

37. \(((x + a) + ((b + z) + z1)) \rightarrow ((x + z) + z1)\)
38. \(((x + (y + a)) + (b + z)) \rightarrow ((x + y) + z)\)

\[ \ldots \]

No rewrite rule in R2.
On this example, the second completion method is less interesting than the first one, since it does not terminate. This example illustrates the difference of power between the two completion methods.

4.3. Arithmetic theory

This third example is again a case where $\rightarrow_{\omega,N,E}$ rewriting can be usefully chosen. Let + and * be addition and multiplication declared as associative-commutative, $s$ be the successor function and $\star$ the exponentiation function. Hullot [13] proposed a rewriting system which has the Church-Rosser property:

\[
\begin{align*}
(0 + x) & \rightarrow x \\
(0 \cdot x) & \rightarrow 0 \\
(s(x) + y) & \rightarrow s((x + y)) \\
(s(x) \cdot y) & \rightarrow ((x \cdot y) + y) \\
(x \cdot (y + z)) & \rightarrow ((x \cdot y) + (x \cdot z)) \\
(x \star 0) & \rightarrow s(0) \\
(s(0) \star x) & \rightarrow s(0) \\
(x \star s(x)) & \rightarrow (x \star s(x)) \\
(x \star (y + z)) & \rightarrow ((x \star y) \cdot (x \star z)) \\
((x \star y) \cdot (z \star y)) & \rightarrow ((x \star z) \star y)
\end{align*}
\]

REVEUR-3 with the rewriting strategy $\rightarrow_{\omega,N,E}$ terminates with the message:

You are currently working modulo the following axioms:

\[
\begin{align*}
((x + y) + z) & \equiv (x + (y + z)) \\
(x + y) & \equiv (y + x) \\
((x \cdot y) \cdot z) & \equiv (x \cdot (y \cdot z)) \\
(x \cdot y) & \equiv (y \cdot x)
\end{align*}
\]

Rewrite rules in R1:

\[
\begin{align*}
1. \quad (0 + x) & \rightarrow x \\
2. \quad (0 \cdot x) & \rightarrow 0 \\
3. \quad (x + 0) & \rightarrow x \\
4. \quad (x \cdot 0) & \rightarrow 0 \\
5. \quad (x \star 0) & \rightarrow s(0) \\
6. \quad (s(0)) \star x & \rightarrow s(0) \\
7. \quad (s(x) + y) & \rightarrow s((x + y)) \\
8. \quad (y + s(x)) & \rightarrow s((x + y)) \\
9. \quad (s(x) \cdot y) & \rightarrow ((x \cdot y) + y) \\
10. \quad (y \cdot s(x)) & \rightarrow ((x \cdot y) + y) \\
11. \quad (x \star s(y)) & \rightarrow (x \star (x \star y)) \\
12. \quad (x \cdot (y + z)) & \rightarrow ((x \cdot y) + (x \cdot z)) \\
13. \quad ((y + z) \cdot x) & \rightarrow ((x \cdot y) + (x \cdot z)) \\
14. \quad (x \star (y + z)) & \rightarrow ((x \star y) \cdot (x \star z))
\end{align*}
\]
Rewrite rules in R2:

15. \(((x \ast y) \ast (z \ast y)) \rightarrow ((x \ast z) \ast y)\)
   Which has for extensions:
16. \(((x \ast y) \ast (z1 \ast y)) \ast z) \rightarrow (((x \ast z1) \ast y) \ast z)\)

Your system is complete!

4.4. Axiomatization of equality and ordering

Let us assume that + and * are associative-commutative symbols and that = is a commutative one. ∧ means the conjunction and is also associative-commutative. ≼ is an ordering relation. The following equations are given to the system:

\[
\begin{align*}
(x \land \text{true}) &= x \\
(x + 0) &= x \\
(x \ast 0) &= 0 \\
(x \ast 1) &= x \\
((x + u) = (y + u)) &= (x = y) \\
(x = x) &= \text{true} \\
(x ≤ (x + u)) &= \text{true} \\
(x \ast (x + z)) &= ((x \ast y) + (x \ast z))
\end{align*}
\]

Using \(\rightarrow_{\text{LUCF}}\) as rewriting relation, we get the following message, where all the equations containing a variable \(v\) have been discovered by the system:

Your are currently working modulo the following axioms:

\[
\begin{align*}
((x \ast y) \ast z) &= (x \ast (y \ast z)) \\
(x \ast y) &= (y \ast x) \\
((x + y) + z) &= (x + (y + z)) \\
(x + y) &= (y + x) \\
((x \ast y) \ast z) &= (x \ast (y \ast z)) \\
(x \land y) &= (y \land x) \\
(x = y) &= (y = x)
\end{align*}
\]

No user equation.

No critical pair equations.

Rewrite rules in R1:

1. \((x \land \text{true}) \rightarrow x\)
2. \((x + 0) \rightarrow x\)
3. \((x \ast 0) \rightarrow 0\)
4. \((x \ast 1) \rightarrow x\)
5. \((\text{true} \ast v1) \rightarrow v1\)
6. \((0 + v2) \rightarrow v2\)
7. \((0 \ast v3) \rightarrow 0\)
5. Conclusion

Among several untyped completion procedures that have been designed during these last years, let us mention KB [13, 6], RRL [18] and the work of Pedersen [31]. Each of this system has its own originality.

REVEUR-3 is an attempt to increase their power in three different ways that are worth emphasizing in this conclusion.

The underlying theoretical approach both unifies different known completion processes and improve them, in the sense that the general completion algorithm implemented in REVEUR-3 is able to deal with a larger class of equational theories: remind that all the linearity conditions introduced by Huet or by Peterson and Stickel have been dropped in our approach. The class of allowed built-in theories is thus largely extended.

Moreover, REVEUR-3 allows to mix commutative and associative-commutative operators with other ones without properties, which was not allowed in other implementations of completion procedures.

The parameterization of REVEUR-3 allows making clear that there is no unique method to complete a set of equations as soon as there is a built-in theory. Some experiments have been performed with associative-commutative built-in theories and do not provide unknown results about decidability of equational theories. Nevertheless our contribution in this case was to provide a more efficient rewriting method for example for abelian groups and arithmetic theories. ERIL [5] is another completion system using the same idea of parameterization for order-sorted algebras without built-in theories.

The power of REVEUR-3 will be further increased in the next future by several enhancements in different directions:

- REVEUR-3 has been designed in order to support any built-in theory for which algorithms for equality decision, matching and unification are known. The next developments of REVE will include such implementations. Among them let us
mention for instance the minus theory [20] defined by the following axioms:

\[-(-x) = x, \quad -(f(x, y)) = f(-y, -x)\]

and the permutative theory [14, 19] defined by

\[f(f(x, y), z) = f(f(x, z), y).\]

- Mixing rewriting relations can be extended to other cases, along the lines of [22].
- Improvements of efficiency are to be considered as well in the rewriting process as in the computations of critical pairs, following for instance [24].
- An increased power would result from the implementation of automatic or semiautomatic ordering for equational rewriting systems.

The modularity of the system makes it easy to modify to deal with other functionalities. The completion process is the basis of a theorem prover in equational logic. In addition to prove equational theorems, it can be used to perform proofs by induction in initial algebras [21] and to solve equations in equational theories [33]. REVEUR-3 can thus be considered as a fundamental tool for proving correctness of programs written for instance in OBJ [8] or LARCH [10].

Acknowledgment

We sincerely thank Jean-Pierre Jouannaud, Pierre Lescanne and Jean-Luc Rémy for their comments about a preliminary version of this paper and Jalil Mzali for his implementation of the pattern matching algorithms in REVEUR-3.

References


