ASTRE: a transformation system using completion

Francoise Bellegarde

Oregon Graduate Institute
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 91-017

August, 1991

ASTRE: a transformation system using completion

Françoise BELLEGARDE *
Western Washington University
Computer Science Department
Bellingham WA. 97226
email: bellegar@strawberry.cs.wwu.edu

Abstract

ASTRE is a program transformation system whose central parts are partial completion procedures. A description of ASTRE and scripts of many examples show that such a system allows a minimum of intervention from the user and provides all the theorem proving abilities that are required for a good transformation system.

1 Program transformation and rewriting

Dershowitz [7, 9] has shown how the completion procedure can be applied to program transformation by automatizing the instanciation, folding and unfolding processes originated by Burstall and Darlington [4]. But no attempt has been made to apply the completion to design a program transformation system.

We think that program transformation can take its place in software design only if the transformation process is automated as much as possible. Moreover the program transformation process requires theorem proving abilities such as ground convergence proofs, induction proofs, sufficient completness, that can be easily provided by completion procedures and other term rewriting system techniques.

The system Focus [21, 22] uses rewriting and overlappings but it does not use completion procedures. The main reason is that a completion procedure is doing too much for the purpose of the transformation. That is why ASTRE limits the completion process and uses "partial completion procedures". Justification of this point of view has been done in [3]. In this paper, our goal is to present ASTRE and examples of program transformation using ASTRE.

2 Basic notions and notations

A program is presented by a set of equalities (or equations). An operational semantics can be given to functions defined by equations by using term rewriting systems.

An equation is a pair of terms $s = t$. We will denote by $T(\Sigma, X)$ the set of terms built with the variables X and the functions symbols of the signature Σ. $V(t)$ denotes the set of the variables of t. The set of ground terms or terms without variables is denoted by $T(\Sigma)$.

*This research was conducted at the Oregon Graduate Institute of Science and Technology, Beaverton, OR 97005

Given a binary relation, $\rightarrow$, $\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$. $\leftrightarrow^*$ is its reflexive and symmetric transitive closure. A relation $\rightarrow$ is noetherian or (strongly) terminating if there is no infinite sequence $t_1 \rightarrow t_2 \cdots$. A relation $\rightarrow$ is confluent if $\leftarrow^* \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$, where $\circ$ denotes the composition of relations.

t/p is the subterm of t at the position p. The replacement of the subterm t/p in t by the term u is denoted by $t[p \leftarrow u]$.

A rule is an oriented pair of terms $l \rightarrow r$. We must have $V(r) \subseteq V(l)$. A term rewriting system is a set of rules. Given a term rewriting system R, the rewriting relation $\rightarrow_R$ is a binary relation in $T(\Sigma, X)$. $s \rightarrow_R t$ if there exists a rule $l \rightarrow r$ in R, a position p in s, a substitution σ such that $\sigma(l) = s/p$ and $t = s[p < -\sigma(r)]$. A term t is in normal form if it is irreducible.

A term rewriting system is terminating if the relation $\rightarrow_R$ is noetherian, confluent if the relation $\rightarrow_R$ is confluent, and convergent if it is both confluent and terminating. Convergence ensures existence and unicity of the normal form of every term.

Critical pairs are produced by overlaps of two redexes in a same term. A non-variable term t' and a term t overlap if there exists a non-variable position p in t such that t/p and t' are unifiable. Let $g \rightarrow d$ and $l \rightarrow r$ be two rules such that l and g overlap at the position p with the most general unifier σ. The overlapped term $\sigma(g)$ produces the critical pair (p, q) defined by $p = \sigma(g[p < -r])$ and $q = \sigma(d)$. A critical pair is convergent if p and q reduce to the same term.

The *completion procedure* [14] was introduced as a means at deriving convergent term-rewriting systems used as procedures for deciding the validity of identities (the word problem) in a given equational theory. The procedure generates new rewrite rules to resolve ambiguities resulting from existing rules that overlap. These new rules are produced by non-convergent critical pairs.

A completion procedure can fail because it is unable to orient an equation into a rule without losing the termination property of the system. However, non-orientable equations may sometimes be used for reduction anyway, because their instances can be oriented. This idea is basic to the unfailing completion procedure [2, 1]. It uses the notion of ordered rewriting which does not require that an equation always be used from left to right. An ordered rewriting system is a set of equations together with a reduction ordering $>$, i.e. a well-founded, monotonic and stable. An ordered rewriting system can be denoted $(E, >)$. When the equations in E can be oriented with $>$, we usually call them rules. The ordered rewriting relation using $(E, >)$ is the rewriting relation $\rightarrow_{E>}$ where $E >$ denotes the set of all the orientable instances of E. This allows us to extend the notion of critical pairs to ordered critical pairs and to extend the completion process to an unfailing completion process, i.e. a completion that cannot fail. The outcome of the unfailing completion procedure, when it does not loop, is either a (ground) convergent term rewriting system R when all equations are rules or a ground convergent ordered rewriting system $(E, >)$ when some equations remain unordered. By *ground convergence*, we mean termination and confluence on ground terms. Obviously, convergence implies ground convergence.

Given a ground convergent term rewriting system R, a term t is *ground (or inductively) reducible* with R if all its ground instances are R reducible.

An equation $s = t$ is an *inductive theorem (or inductive consequence)* of E if for any ground substitution σ, $\sigma(s) = \sigma(t)$.

We consider programs presented in a specification $S = (\Sigma, E)$ by a set of equations E. We consider the case when the set of functions in the signature Σ can be split into a set of constructors C and a set of defined functions D. The definition of functions of D

is *sufficiently complete with respect to C*, i.e. it produces no junk, if every ground term is provably equal to a *constructor term*, which is a term built only with constructors.

In its operational point of view, the program is presented by a term rewriting system R for the specification (Σ, R). Computation of a (ground) term in $T(\Sigma)$ is done by rewriting. The operationally complete definition of a function f w.r.t C is when for all ground term $f(t_1, \cdots, t_n)$, there exists a constructor term s such that

$$f(t_1, \cdots, t_n) \rightarrow_R^* s$$

3 ASTRE, a system for program transformation

ASTRE has been written in CAML [10]. The implementation uses largely all the functions CAML provided by the system ORME [18, 19]

A functional language like CAML can be considered as a good specification language and it is easy to translate a program written in a pure functional language into a set of first order equations. Let us consider a simple CAML program to computes permutations which can be found in [12]:

Example 1

```
let rec permut =
  let perms l x = map (cons x) (permut (discard(x l))
                  where rec discard x = function
                        []->[]
                        |y::ys -> if (x=y) then ys
                                          else y::discard(x,ys)
    in function [] -> [[]]
                |l -> flatten (map(perms l) l)
                      where rec flatten = function
                       [] -> []
                       |x::xs -> x @ flatten(xs);;
```

By transforming the higher order functions, it can be written as a set of first-order equations:

```
discard(x,[])=[]
discard(x,y::ys)= if (x=y) then ys
                           else  y::discard(x,ys)
perms(l,x)= map_cons(x,permut(discard(x,l)))
permut(l)=if null(l) then [[]]
                     else flatten(map_perms(l,l))
```

ASTRE considers a program of a function f presented by the set of equalities E in the specification $S = (\Sigma, E)$. f is a distinguished function in the set of function symbols of Σ we call *main symbol* for the transformation. ASTRE considers the case when the set of functions in the signature Σ can be split into a set of *constructors* C and a set of *defined functions* D. ASTRE transforms the program of f presented by the set of equalities E into a program of f presented by a set of equalities E'.

The presentation of a program by a set of equalities E is a term rewriting system R whose rules are the equalities in E directed from left to right. For example, the program for permutations is the following term rewriting system:

```
discard(x,[]) -> []
discard(x,y::ys) -> if (x=y) then ys
                            else  y::discard(x,ys)
perms(l,x) -> map_cons(x,permut(discard(x,l)))
permut(l) -> if null(l) then [[]]
                    else flatten(map_perms(l,l))
```

The result of computation of a (ground) term like *permut*([1; 2; 3]) is obtained by (ground) rewriting in the (ground) normal form:

$$[[1;2;3];[1;3;2];[2;1;3];[2;3;1];[3;1;2];[3;2;1]].$$

The correctness of the transformation is ensured when R and R' are *ground convergent.* Weaker sufficient conditions for the correctness of the transformation involve properties such as the (operationally) sufficient completness of the specifications S and S' [3]. ASTRE can verify the ground convergence of R and R' by using an unfailing completion procedure.

The transformation process also involves a (partial) unfailing completion procedure. The original code of an unfailing completion procedure is borrowed from the system ORME. It is described in [18, 19].

For the transformation, the user provides definitions of new function symbols.

Example 2

For example, a transformation of *permut* can be initiated by introducing the new definition:

```
fmp(x,y)=flatten(map_perms(x,y).
```

The result of the transformation process given by ASTRE is then:

```
[<<permut(x1)->IF(null(x1),[[]],fmp(x1,x1))>>;
<<discard(x1,x2::x3))->IF((x1=x2),x3,x2::discard(x1,x3)))>>;
<<discard(x1,[])->[]>>;
<<map_cons(x1,x2::x3))->(x1::x2)::map_cons(x1,x3))>>;
<<map_cons(x1,[])->[]>>;
<<fmp(x1,[])->[]>>;
<<fmp(x1,(x2::x3))
    ->
    (map_cons(x2,IF(null(discard(x2,x1)),[[]] ,
    fmp(discard(x2,x1),discard(x2,x1)))) @ fmp(x1,x3))
    >>]
```

This corresponds to the CAML program:

```
let permut = function
               []->[[]]
                l->
                   let rec discard x = function
                         []->[]
                         |y::ys -> if (x=y) then ys
                                             else y::discard(x,ys)
                in let fmp(x,y) = match y with [] -> []
                     |y::ys -> (map (cons y) (if null(discard y x))
```

```
                                then [□]
                                else fmp(discard(y,x),discard(y,x))))
                                @ fmp(x,ys)
        in fmp(1,1);;
```

The transformation process gives a direct recursive definition of the function *fmp*.

Most of the time, properties of functions are required for the simplification of the definition. For the example above, the transformation will give

```
<<fmp(x,□) -> □ @ □>>
```

if the user does not supply the axiom:

```
□ @ x = x
```

in the middle of the transformation process.

In ASTRE, the user can provide these properties interractively at any time during the transformation process which is very useful for simplification properties. In fact, ASTRE asks the user:

```
ADD A LEMMA? ANSWER YES or NO.
```

at each new loop of the completion process.

Moreover, properties of functions are often required for the synthesis of new recursive definitions as shown by the following example:

Example 3

```
The input for the transformation is:
<<CONC_ALL(NIL) -> NIL>>
<<CONC_ALL(C(x,xs)) ->(ALL(x) @ CONC_ALL(xs))>>
<<FILTER(NIL) -> NIL>>
<<FILTER(C(x,xs)) -> IF(ISSIG(x),C(x,FILTER(xs)),FILTER(xs))>>
<<SIGPERM(x) -> FILTER(CONC_ALL(x))>>

,and the axiom given by the user is:

<<FILTER(x @ y) = (FILTER(x) @ FILTER(y))>>

,and the definition of a new symbol is:

[<<FILTER(CONC_ALL(x))=CONCFIL(x)>>]

the result of the transformation is:

[1  <<CONCFIL(NIL)->NIL>>;
2  <<FILTER(C(x1,x2))->IF(ISSIG(x1),C(x1,FILTER(x2)),FILTER(x2))>>;
3  <<SIGPERM(x1)->CONCFIL(x1)>>;
4  <<FILTER(NIL)->NIL>>;
5  <<CONCFIL(C(x1,x2))->(FILTER(ALL(x1)) @ CONCFIL(x2))>>]
```

```
without the axiom about the endomorphism of FILTER,
the result of the transformation would have been:

[1  <<CONCFIL(C(x1,x2))->FILTER((ALL(x1) @ CONC_ALL(x2)))>>;
2  <<SIGPERM(x1)->CONCFIL(x1)>>;
3  <<CONC_ALL(NIL)->NIL>>;
4  <<CONCFIL(NIL)->NIL>>;
5  <<FILTER(NIL)->NIL>>;
6  <<CONC_ALL(C(x1,x2))->(ALL(x1) @ CONC_ALL(x2))>>;
7  <<FILTER(C(x1,x2))->IF(ISSIG(x1),C(x1,FILTER(x2)),FILTER(x2))>>]
```

Generally, the transformation process as well as the verification of the ground convergence of the programs relies upon an unfailing completion procedure.

4 Ground Convergence

The verification of the ground convergence of the program is offered as an option to the user. ASTRE uses the ER-COMPLETION procedure implemented in ORME for the verification of ground convergence. It is described in [18, 19].

It requires a *complete reduction ordering* i.e. an ordering that is total on ground terms. This condition is fulfilled by an ordering based on polynomial interpretations [17, 5], called *polynomial ordering*, or by a *recursive path ordering* [6] based upon a precedence which is total on function symbols. These two kind of complete reduction orderings are available in ASTRE.

When the user choses the polynomial ordering, he must provide the polynomial interpretations before entering in the verification of the ground convergence.

When the user choses the recursive path ordering, the total precedence can be provided interractively during the completion process. Recall that ASTRE requires from the user the list of the constructor symbols. This allows us to get automatically an initialization of the total precedence by putting the constructors at the bottom of the precedence. This way, there are few interractions with the user.

Moreover, the programs are mostly presented as *constructor systems.*

Definition 1 *A constructor system is a set of constructor equations. Constructor equations are of the form:* $f(t_1, \cdots, t_n) = t$ *where* $t \in T(F,V)$, $f \in D$, $t_i, i = 1, n \in T(C)$. *These equations must be oriented as (constructor) rules of a term rewriting system from left to right.*

When the program is presented by a constructor system, a total precedence of the symbols for the recursive path ordering is derived automatically by ASTRE. This is provided by the option *"automatique rpo"*. You can find in annexe 9.1 the script of the verification of the ground convergence of the presentation of the program for the example 1. The "automatique rpo" can fail because it finds a circuit in the precedence it attempts to generate, It is the case for the program for permutations given in example 1. A script is given in annexe 9.2. It is easy to see that the program for permutations is not strongly terminating.

5 UNFAIL ASTRE

The transformation process is based on a partial unfailing completion. Ordered critical pairs are computed between the new definitions provided by the user and the program presented by the set of (ground convergent) rules R. Simplifications by rewriting are performed. The axioms provided by the user at any time during the completion process are overlapped with the rules of R and are used for simplification.

5.1 Orderings

Once again, a complete reduction ordering must be provided to the completion procedure which uses it for simplification and computation of ordered critical pairs. The transformation is obtained by the ordered critical pairs between the definition of a new symbol and the program presented by the system R.

The orientation of the rules that is required for getting the desired overlaps is contradictory with the "automatic rpo" we defined for the verification of the ground convergence in Section 4. For the example 2, the definition $FILTER(CONC\_ALL(x)) = CONCFIL(x)$ is oriented from right to left by the "automatique rpo". However it is the opposite orientation that brings overlaps with the definition of $CONC\_ALL$. The critical pairs coming from these overlaps leads to the desired recursive and complete definition of $CONCFIL$.

A recursive path ordering or a polynomial ordering are available in option. They are used in the transformation step in the examples 9.3 and 9.4 in annexe. The advantadge of the recursive path ordering is its incrementality. The user can define the precedence along the verification of ground convergence and he can continue during the transformation process. It is relatively easy to find a total precedence when the recursive path ordering works but it can be more difficult to find polynomial interpretations. Moreover, it is difficult for the user to take care of the definition of the ordering before or during the transformation process, and these two orderings are largely insufficient for the purpose of the transformation process.

Hopefully, the user knows how to orient equations to get the desired overlaps or the desired simplification. He can give directions on how to orient the equations along with the transformation process under *the option "manual orientation"*. So doing the user must be careful not to introduce possibilities of loops in rewriting. The system could try to infer a recursive path ordering from directions given by the user but this is not yet implemented in ASTRE.

5.2 Termination of the transformation step

An unfailing completion does not fail but it can loops. However, in this case, the transformation is done before the completion process terminates.

A transformation step can be considered as terminated when the completion procedure has generated:

1. a recursive and operationally complete definition of the new symbol introduced by the user,
2. and an operationally complete definition of the main symbol invoking this new symbol,
3. and when these definitions are enough simplifed.

Obviously, the last requirement depends only on the judgement of the user. The completion process may continue to generate equational consequences (critical pairs) which may eventually simplify the definitions. If the definitions of the symbols in Σ are operationally complete and if the orientation is well chosen, an operationally complete definition of the new symbol will be automatically generated by the completion process. However, this definition is useful for the transformation only if it is recursive and if this simplifies the definition of the other symbols. We have seen that appropriate axioms can be supplied to help fulfilling these goals. There can be two cases, either the transformation will not succeed to reach these goals, or it will, if we wait enough to let it generates the required equations. The requirements 1 and 2 above could be automatically checked using a test of sufficient completness citeKapur-Narendran-Zhang87,Kounalis90. But the user must finally decide whether or not to quit the process.

5.3 User's tasks

At this point, we can summarize the role of the user during the transformation process. The user must:

- provide the axioms that are necessary to simplify the definitions and get a recursive definition of the new symbol,
- chose the orientation of the equations to get the right overlaps and the right simplifications,
- and, eventually, decide to quit the completion process.

This requires a good knowledge about the properties of the functions invoked and a clear goal for the transformation and it is not always obvious. The transformation process must be flexible enough to allow the user to undo. Sometimes the user may have some doubts about the orientation of an equation $E1$ simply because the transformation has not yet generated the equation $E2$ which could be subsumed or simplified by $E1$ oriented in a certain direction. When asked for the orientation of $E1$, the user can let it undefined. However, when he understands how to orient it for the purpose of the transformation, he then can interrupt temporarily the completion process to ask for considering the orientation of the equations that has been let undefined. He can also through away some equations that seems to be useless if he judges that the presence of these equations will slow down the transformation process by generating more useless equational consequences. These features gives the user full control of the transformation process and a lot of flexibility on the degree of partiality of the completion process invoked for the transformation. The counterpart is that it increases the responsability of the user. An example of a transformation using these features is given in annexe 9.5.

5.4 Automatique deletion

Let us now consider the result after the termination of the partial completion process. The axioms added by the user, some of the equations or rules generated during the transformation are not part of the presentation of the program R' result of the transformation. If the transformation succeeds, this program is an operationally complete definition of the main symbol invoking the new symbol which is itself defined by a recursive and operationally

complete definition. The program must be a constructor system. ASTRE eliminates automatically the non constructor rules and the constructor rules that are not invoked in the definition of the main symbol. However, the present implementation of the "automatique deletion" can be insufficient. For the example in annexe 9.5, the automatique deletion let the two rules:

```
<<G(x1,ZERO)->ZERO>>
<<G(ZERO,x1)->x1>>
```

Which one of these two rules is contained in the definition of the function G? The decision can be done by extracting an operationally complete and recursive definition of G. An automatic extraction would use a test of sufficient completness. This is not done automatically in the current implementation. The user must chose the last few rules that need to be deleted.

5.5 Ground Convergence of the result

We know that:

- If the transformation process has been controlled by a recursive path ordering or by a polynomial ordering and if nothing has been deleted, R' is obviously ground convergent.
- If R is ground convergent and if R' give an operationally complete definition of the main symbol, R' is ground convergent by a result proved in [16] and recall in [3]. Therefore, if the deletion is correct, then R' gives an operationally complete definition of the main symbol, and thus, R' is ground convergent.

However, in the absence of a test of sufficient completness, the verification of the ground convergence of R' is given as an option to the user.

At the end of the transformation step, the user is asked for another transformation step. The examples in annexes 9.6 and 9.9 three transformation steps. ASTRE can be used to transform large programs.

6 AC ASTRE

Let us consider the particular case when the transformation requires the associativity and the commutativity axioms of some function symbols.

Example 4

For example let suppose that we have a function definition DOT which computes $\Sigma_{i=(1,n)}(x_i * y_i)$ and we use it to define a function $SUMDOT$ which computes $\Sigma_{i=(1,n)}(x_i * y_i) + \Sigma_{i=(1,n)}(u_i * v_i)$. The presentation of the program is:

```
<<DOT(x,y,ZERO)=ZERO>>
<<DOT(x,y,S(n))=(DOT(x,y,n) + (I(x,S(n))*I(y,S(n))))>>
<<SUMDOT(x,y,u,v,n)=(DOT(x,y,n)+DOT(u,v,n))>>
```

$SUMDOT$ could be computed as $\Sigma_{i=(1,n)}((x_i * y_i) + (u_i * v_i))$. The transformation requires the associativity and the commutativity of $+$.

If we use the UNFAIL ASTRE, 27 equations are generated before the equation which will simplify the definition of SD in a recursive definition (see example in annexe 9.7). For each of these equations, the user will be asked for orientation in the mode "manual ordering". It can be a long and overwhelming task.

ASTRE provides for this case a transformation step based on an associative and commutative completion procedure [20]. The user asks for the mode AC ASTRE. AC ASTRE uses the ANS-AC-completion provided by ORME. The competion procedure uses associative and commutative unification algorithm in place of the syntactic unification algorithm to generate critical pairs. Associative and commutative matching is used to apply rules. For the example of $SUMDOT$, AC ASTRE gives the result directly (see example in annexe 9.8).

There is some drawbacks in using an associative and commutative completion procedure. It is not an unfailing completion and therefore all equations must be oriented. It is less flexible. A way out is given by AC ASTRE which provides to the user a garbage to postpone the orientation of some equations or to through away unorientable equations. So doing, we get back the flexibility. Another point is that the orderings that are applicable to associative and commutative rewriting are few. Polynomial ordering is compatible with associativity and commutativity [5] but the recursive path ordering is not. Practically, it is not very important because we mostly use the option "manual orientation" for the transformation process.

7 Utilisation of ASTRE

Let us consider an example. We want a program to compute the list of the prefixes of a given list.

$$PREFIXES([1;2;3;4]) = [[];[[1]];[[1];[2]];[[1];[2];[3]];[[1];[2];[3];[4]]]$$

The programmer write the following CAML program:

```
(*|
Value REPEATL : ('a list -> 'a list list)
 CAML_system{hd}
REPEATL([1;2;3;4])=[[1];[1];[1];[1]]
|*)
let REPEATL(l) =
        let rec REP = function
              (l,[])->[]
              |(l,x::xs)->[hd(l)]::REP(l,xs)
        in REP(l,l);;
(*|
Value ZIP : ('a list * 'b list -> ('a * 'b) list)
ZIP([1;2;3;4];[5;6;7;8])=[(1,5);(2,6);(3,7);(4,9)]
|*)
let rec ZIP=function
         ([],[])->[]
        |(x::xs),(y::ys)->(x,y)::ZIP(xs,ys);;
(*|
Value CONS_NIL : ('a list list -> 'a list list)
```

```
|*)
let CONS_NIL x = []::x;;
(*|
Value PREFIXES : ('a list -> 'a list list list)
{CONS_NIL,REPEATL,ZIP},
 CAML_system{map,o}
|*)
let rec PREFIXES = function
          []->[[]]
         |x::xs-> let CONS=prefix ::
                in let PREF=CONS_NIL o (map CONS) o ZIP
           in PREF(REPEATL(x::xs),PREFIXES(xs));;
```

The presentation of this program by a system of first order equations is:

```
<<REPEATL(l)=REP(l,l)>>
<<REP(l,NIL)=NIL>>
<<REP(l,C(x,xs))=C(C(HD(l),NIL),REP(l,xs))>>
<<PREF(x,y)=CONS_NIL(MAP_CONS(ZIP(x,y)))>>
<<CONS_NIL(x)=C(NIL,x)>>
<<ZIP(NIL,NIL)=NIL>>
<<ZIP(C(x,xs),C(y,ys))=C(PAIR(x,y),ZIP(xs,ys))>>
<<MAP_CONS(NIL)=NIL>>
<<MAP_CONS(C(PAIR(x,y),l))=C(C(x,y),MAP_CONS(l))>>
<<PREFIXES(NIL)=C(NIL,NIL)>>
<<PREFIXES(C(x,xs))=PREF(REPEATL(C(x,xs)),PREFIXES(xs))>>
```

ASTRE verifies the ground convergence by "automatique rpo". We can transform the definition of $PREF$ by introducing a new symbol PP for the composition (MAP_CONS o ZIP). This yields to a first transformation step:

```
PREFIXES TRANSFORMATION
Eureka_rules
[<<MAP_CONS(ZIP(x1,x2))->PP(x1,x2)>>]
ADD A LEMMA?? ANSWER YES or RETURN
<<C(C(x1,x2),PP(x3,x4)) = PP(C(x1,x3),C(x2,x4))>>
ANSWER PLEASE > < UNDEF or QUIT
<
ADD A LEMMA?? ANSWER YES or RETURN
<<PP(NIL,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
AUTOMATIQUE DELETION
[1  <<PP(NIL,NIL)->NIL>>;
2  <<PREFIXES(C(x1,x2))
   ->C(NIL,PP(C(C(HD(C(x1,x2)),NIL),REP(C(x1,x2),x2)),PREFIXES(x2)))>>;
3  <<REP(x1,C(x3,x2))->C(C(HD(x1),NIL),REP(x1,x2))>>;
4  <<REP(x1,NIL)->NIL>>;
5  <<PREFIXES(NIL)->C(NIL,NIL)>>;
6  <<PP(C(x1,x3),C(x2,x4))->C(C(x1,x2),PP(x3,x4))>>]
```

We can now slightly generalize the subterm $REP(C(x1,x2),x2))$ in the right-hand side of the definition of $PREFIXES(C(x1,x2))$. We introduce the new symbol G for

$$REP(C(x,y),z) = G(x,z).$$

The variable y disappears in the left-hand side. The user can be aware that the tail y is useless in the definition of REP. This yields to a second transformation step:

```
PREFIXES TRANSFORMATION
Eureka_rules
[<<REP(C(x1,x2),x3)->G(x1,x3)>>]
ADD A LEMMA?? ANSWER YES or RETURN
<<G(x1,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
<<G(x1,C(x2,x3)) = C(C(HD(C(x1,x4)),NIL),G(x1,x3))>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
YES
<<HD(C(x1,x2))=x1>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
AUTOMATIQUE DELETION
[1  <<G(x1,NIL)->NIL>>;
2  <<PREFIXES(C(x1,x2))->C(NIL,PP(C(C(x1,NIL),G(x1,x2)),PREFIXES(x2)))>>;
3  <<PP(NIL,NIL)->NIL>>;
4  <<PREFIXES(NIL)->C(NIL,NIL)>>;
5  <<PP(C(x1,x3),C(x2,x4))->C(C(x1,x2),PP(x3,x4))>>;
6  <<G(x1,C(x2,x3))->C(C(x1,NIL),G(x1,x3))>>]
```

Recall that in a term rewriting system, for a rule $l \rightarrow r$, we must have $V(r) \subseteq V(l)$ (see Section 2). In the middle of the transformation, the user provides the axiom $HD(C(x,y)) = x$ which allows the correct orientation of the definition of $G(x1,C(x2,x3))$. Now the program does not invoke the function HD.

The definition of $PREFIXES$ can still be improved. The subterm

$$C(C(x1,NIL),G(x1,x2))$$

is nothing else but an unfolding of $G(x1,C(x1,x2))$. We can try a generalization of

$$PP(G(x1,C(x1,x2)),PREFIXES(x2))$$

We introduce the new symbol K for $PP(G(x,u),v) = K(x,v)$. The variable u does not occur in the right-hand side. The script of the corresponding third transformation step is in annexe 9.9. The user quits the completion when a complete definition of K is generated. After (automatique) deletion, the result is:

```
[1  <<PREFIXES(NIL)->C(NIL,NIL)>>;
2  <<PREFIXES(C(x1,x2))->C(NIL,K(x1,PREFIXES(x2)))>>;
3  <<K(x1,NIL)->NIL>>;
4  <<K(x1,C(x2,x3))->C(C(C(x1,NIL),x2),K(x1,x3))>>]
```

This corresponds to the short CAML program:

```
let rec PREFIXES=function
         []->[[]]
        |x::xs->  let rec K = function
                    (x,[])->[]
                   |(x,y::ys)-> ([x]::y)::K(x,ys)
                in []::K(x,PREFIXES(xs));;
```

8 Conclusion

ASTRE shows that completion procedures can be basis of reasonable transformation systems. However, work about orderings compatible with the transformation process would be useful.

ASTRE can be improved. It could ensure more theorem proving abilities such as inductive proofs [23, 16] for the inductive axioms the user provides during the completion process or test for sufficient completness [13, 15] for proving the correctness of the transformation. This test could allows ASTRE to advise the user when the transformation step is possibly terminated.

Our goal is to minimalize the intervention of the user. We could improve ASTRE by providing a "replay" like in the system Focus [22]. The idea is to automatize completely a transformation which acts like a previous one. For this purpose, more work in studying strategies for transformations will be helpful. These strategies can be derived from the theorems on higher order functions like $map\ f \circ map\ g = map(f \circ g)$.

ASTRE is restricted to equational specifications. We could consider conditional specifications [11, 24]. Even in the case of an equational specification, conditional rewriting would be particularly useful when we want to introduce cases for the transformation.

I would like to thank Dick Kieburz, Claude Kirchner, Helene Kirchner, Pierre Lescanne, who provided me with encouragements and support, Pierre Lescanne who gave me access to the system ORME, Dick Kieburz for wise advices and examples.

9 Annexes

9.1 Ground Convergence by automatique rpo

Here follows the script of the verification of the ground convergence of an initial program for the example 2:

```
GETTING THE SYSTEM
<<CONC_ALL(NIL) = NIL>>
<<CONC_ALL(C(x,xs))=(ALL(x) @ CONC_ALL(xs))>>
<<FILTER(NIL)=NIL>>
<<FILTER(C(x,xs)) = IF(ISSIG(x),C(x,FILTER(xs)),FILTER(xs))>>
<<SIGPERM(x) = FILTER(CONC_ALL(x))>>
```

```
GETTING THE CONSTRUCTORS
C NIL
GROUND CONVERGENCE
ORDERINGS
<1> : incremental_rpo
<2> : manual
<3> : automatique rpo
<4> : polynomial
type 1 or 2 or 3 or 4
3
WARNING
@ is not a constructor or a defined symbol
WARNING
ALL is not a constructor or a defined symbol
WARNING
IF is not a constructor or a defined symbol
WARNING
ISSIG is not a constructor or a defined symbol
======================
Successful Completion
======================
EQUATIONS
□
RULES
[1  <<CONC_ALL(C(x1,x2))->(ALL(x1) @ CONC_ALL(x2))>>;
2  <<FILTER(NIL)->NIL>>;
3  <<CONC_ALL(NIL)->NIL>>;
4  <<SIGPERM(x1)->FILTER(CONC_ALL(x1))>>;
5  <<FILTER(C(x1,x2))->IF(ISSIG(x1),C(x1,FILTER(x2)),FILTER(x2))>>]
```

9.2 Failure of the automatique rpo

Here follows the script of an attempt to verify the ground convergence which fails with a circuit in the total precedence infered by the option "automatique rpo":

```
GETTING THE SYSTEM
<<map_cons(x,NIL)=NIL>>
<<map_cons(x,C(y,ys))=C(C(x,y),map_cons(x,ys))>>
<<discard(x,NIL)=NIL>>
<<discard(x,C(y,ys))=IF(eq(x,y),ys,C(y,discard(x,ys)))>>
<<flatten(NIL)=NIL>>
<<flatten(C(x,xs))= (x @ flatten(xs))>>
<<permut(l)=IF(null(l),C(NIL,NIL),flatten((map_perms (l,l))))>>
<<perms(x,l)=map_cons(x,permut(discard(x,l)))>>
<<map_perms(l1,NIL) =C(NIL,NIL)>>
<<map_perms(l1,C(x,xs))=C(perms(x,l1),(map_perms(l1,xs)))>>
GETTING THE CONSTRUCTORS
C NIL
GROUND CONVERGENCE
```

```
ORDERINGS
<1> : incremental_rpo
<2> : manual
<3> : automatique rpo
<4> : polynomial
type 1 or 2 or 3 or 4
3
The ordering rpo for constructor system gives a circuit in the precedence:
Ground convergence is not verified.
Your equations are considered as rules.
```

9.3 Transformation by recursive path ordering

Here follows the sript of a transformation using a recursive path ordering for the ground convergent system:

```
[1  <<L(C(x2,x1))->(UN + L(x1))>>;
2  <<L(NIL)->ZERO>>;
3  <<(NIL @ x1)->x1>>;
4  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>]
```

The recursive path ordering used to verify the ground convergence can be incremented for the transformation step: The new definition is given by the user. Then, it is oriented by the recursive path ordering:

```
<<L(x @ y)=LA(x,y)>>
GIVE A PRECEDENCE FOR:  L   LA
ANSWER PLEASE > < =  QUIT
<
<<(x1 @ x2),LA(x1,x2)>>
GIVE A PRECEDENCE FOR:  @   LA
ANSWER PLEASE > < =  QUIT
>
```

The interractions with the user during the transformation step are considerably reduced.

```
LA TRANSFORMATION
Eureka_rules
[<<L((x1 @ x2))->LA(x1,x2)>>]
Added Lemmas
□
you will be asked at each step if you want to add a lemma
ADD A LEMMA?? ANSWER YES or RETURN
EQUATIONS
[1  <<LA(NIL,x1)=L(x1)>>]
RULES
[1  <<(NIL @ x1)->x1>>;
2  <<L(NIL)->ZERO>>;
3  <<L(C(x2,x1))->(UN + L(x1))>>;
4  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
```

```
5  <<L((x1 @ x2))->LA(x1,x2)>>]
EQUATIONS
[1  <<(UN + LA(x1,x2))=LA(C(x3,x1),x2)>>]
RULES
[1  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
2  <<(NIL @ x1)->x1>>;
3  <<L(NIL)->ZERO>>;
4  <<L(C(x2,x1))->(UN + L(x1))>>;
5  <<L((x1 @ x2))->LA(x1,x2)>>;
6  <<LA(NIL,x1)->L(x1)>>]
<<LA(x1,x2),LA(C(x3,x1),x2)>>
GIVE A STATUS FOR  LA
ANSWER PLEASE LR RL QUIT
LR
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
======================
Successful Completion
======================
EQUATIONS
[]
RULES
[1  <<LA(NIL,x1)->L(x1)>>;
2  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
3  <<(NIL @ x1)->x1>>;
4  <<L(NIL)->ZERO>>;
5  <<L(C(x2,x1))->(UN + L(x1))>>;
6  <<L((x1 @ x2))->LA(x1,x2)>>;
7  <<LA(C(x3,x1),x2)->(UN + LA(x1,x2))>>]
RESULT OF THE DELETION
[<<LA(NIL,x1)->L(x1)>>;
<<L(NIL)->ZERO>>;
<<L(C(x2,x1))->(UN + L(x1))>>;
<<LA(C(x3,x1),x2)->(UN + LA(x1,x2))>>]
```

9.4 Transformation by polynomial ordering

Here follows the script of a transformation using a polynomial ordering for the system:

```
[1 <<(ZERO + x) -> x>>;
2 <<(S(x) + y) -> S(x + y)>>;
3 <<D(x) -> (x + x)>>]
```

The axiom for the transformation is:

```
<<(x + S(y))=S(x + y)>>,
```

and the new definition is:

```
<<(x + x) = DD(x)>>
```

The user must give the interpretations of the symbols:

```
+,<<(x*y)+1>> S,<<x+1>> ZERO,<<2>> D,<<(x^2)+2>> DD,<<x^2>>
```

The transformation requires no intervention of the user:

```
D TRANSFORMATION
Eureka_rules
[<<(x1 + x1)->DD(x1)>>]
Added Lemmas
[<<(x1 + S(x2))->S((x1 + x2))>>]
you will be asked at each step if you want to add a lemma
ADD A LEMMA?? ANSWER YES or RETURN
EQUATIONS
[1  <<DD(ZERO)=ZERO>>]
RULES
[1  <<(ZERO + x1)->x1>>;
2  <<(S(x1) + x2)->S((x1 + x2))>>;
3  <<D(x1)->DD(x1)>>;
4  <<(x1 + S(x2))->S((x1 + x2))>>;
5  <<(x1 + x1)->DD(x1)>>]
EQUATIONS
[1  <<S(S(DD(x1)))=DD(S(x1))>>]
RULES
[1  <<(S(x1) + x2)->S((x1 + x2))>>;
2  <<(ZERO + x1)->x1>>;
3  <<D(x1)->DD(x1)>>;
4  <<(x1 + S(x2))->S((x1 + x2))>>;
5  <<(x1 + x1)->DD(x1)>>;
6  <<DD(ZERO)->ZERO>>]
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
======================
Successful Completion
======================
EQUATIONS
[]
RULES
[1  <<DD(ZERO)->ZERO>>;
2  <<(x1 + S(x2))->S((x1 + x2))>>;
3  <<D(x1)->DD(x1)>>;
4  <<(S(x1) + x2)->S((x1 + x2))>>;
5  <<(ZERO + x1)->x1>>;
6  <<(x1 + x1)->DD(x1)>>;
7  <<DD(S(x1))->S(S(DD(x1)))>>]
RESULT OF THE DELETION
[<<DD(ZERO)->ZERO>>;
<<D(x1)->DD(x1)>>;
<<DD(S(x1))->S(S(DD(x1)))>>]
```

9.5 Quit by the user

Here follows the transformation of the program:

```
[1 <<(ZERO + x)->x>>;
2 <<(S(x) + y)->S(x + y)>>;
3 <<(ZERO * x)->ZERO>>;
4 <<(S(x) * y)->((x * y) + y)>>;
5 <<F(ZERO)->S(ZERO)>>;
6 <<F(S(x))->(S(x) * F(x))>>]
```

The transformation process loops and will be interrupted by the user. The user will also let the orientation of an equation undefined till he knows he needs it for simplifying the definition of the main symbol.

```
F TRANSFORMATION
Eureka_rules
[<<(x1 * F(x2))->G(x2,x1)>>]
Added Lemmas
[<<((x1 * x2) * x3)->(x1 * (x2 * x3))>>]
```

We have removed the displays from the script and let only the interaction for orientation by the user.

```
<<G(x1,ZERO) = ZERO>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<G(x1,S(x2)) = (G(x1,x2) + F(x1))>>
ANSWER PLEASE > < UNDEF or QUIT
(* The user is not sure how he wants to direct the above equation *)
UNDEF
<<(x1 * S(ZERO)) = G(ZERO,x1)>>
ANSWER PLEASE > < UNDEF or QUIT
<
<<(x1 * G(x2,S(x2))) = G(S(x2),x1)>>
ANSWER PLEASE > < UNDEF or QUIT
<
(* At this stage the display is:*)
EQUATIONS
[1  <<G(x1,S(x2))=(G(x1,x2) + F(x1))>>;
2  <<(x1 * G(x2,x3))=G(x2,(x1 * x3))>>;
3  <<(x1 * (F(x2) * x3))=(G(x2,x1) * x3)>>]
RULES
[1  <<((x1 * x2) * x3)->(x1 * (x2 * x3))>>;
2  <<F(S(x1))->G(x1,S(x1))>>;
3  <<F(ZERO)->S(ZERO)>>;
4  <<(S(x1) * x2)->((x1 * x2) + x2)>>;
5  <<(ZERO * x1)->ZERO>>;
6  <<(S(x1) + x2)->S((x1 + x2))>>;
7  <<(ZERO + x1)->x1>>;
```

```
8  <<(x1 * F(x2))->G(x2,x1)>>;
9  <<G(x1,ZERO)->ZERO>>;
10  <<G(ZERO,x1)->(x1 * S(ZERO))>>;
11  <<G(S(x2),x1)->(x1 * G(x2,S(x2)))>>]
<<(x1 * G(x2,x3)) = G(x2,(x1 * x3))>>
ANSWER PLEASE > < UNDEF or QUIT
>
(* By orienting the above equation from left to right, a simplification
of the rule 11 in a tail recursive definition can be done:
11  <<G(S(x2),x1)->G(x2,(x1 * S(x2)))>>*)
<<(x1 * (F(x2) * x3)) = (G(x2,x1) * x3)>>
ANSWER PLEASE > < UNDEF or QUIT
(* The user does not know how to orient the above equation *)
UNDEF
ADD A LEMMA?? ANSWER YES or RETURN
YES
Give the Lemma in between parenthesis
(* This lemma allows to simplify the rule 10 *)
<<(x1 * S(ZERO))=x1>>
ANSWER PLEASE > < UNDEF or QUIT
>
(* Now the display is: *)
EQUATIONS
[1  <<G(x1,S(x2))=(G(x1,x2) + F(x1))>>;
2  <<(x1 * (F(x2) * x3))=(G(x2,x1) * x3)>>;
3  <<G(x1,((x2 * x3) + x3))=(G(x1,(x2 * x3)) + G(x1,x3))>>]
RULES
[1  <<(S(x1) * x2)->((x1 * x2) + x2)>>;
2  <<(ZERO * x1)->ZERO>>;
3  <<(S(x1) + x2)->S((x1 + x2))>>;
4  <<(ZERO + x1)->x1>>;
5  <<(x1 * F(x2))->G(x2,x1)>>;
6  <<(x1 * S(ZERO))->x1>>;
7  <<G(ZERO,x1)->(x1 * S(ZERO))>>;
8  <<F(ZERO)->S(ZERO)>>;
9  <<F(S(x1))->G(x1,S(x1))>>;
10  <<((x1 * x2) * x3)->(x1 * (x2 * x3))>>;
11  <<G(x1,ZERO)->ZERO>>;
12  <<G(S(x2),x1)->G(x2,(x1 * S(x2)))>>;
13  <<(x1 * G(x2,x3))->G(x2,(x1 * x3))>>]
<<G(x1,((x2 * x3) + x3)) = (G(x1,(x2 * x3)) + G(x1,x3))>>
ANSWER PLEASE > < UNDEF or QUIT
(* The user could quit the completion
but he wants to reconsider the equation 1*)
QUIT
Do you want to continue the step changing the equations? answer YES or NO
YES
Do you want to delete some equations YES or NO
```

```
YES
Give the numbers of the equations you want to delete
Separate the numbers by a space
2 3
<<G(x1,S(x2)) = (G(x1,x2) + F(x1))>>
ANSWER PLEASE > < UNDEF or QUIT
<
(* The completion generates now a few equations that are useless for
the transformation: *)
<<(x1 * (G(x2,x3) * x4)) = (G(x2,(x1 * x3)) * x4)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<(x1 * ZERO) = G(x2,(x1 * ZERO))>>
ANSWER PLEASE > < UNDEF or QUIT
<
<<(x1 * (x2 * ZERO)) = G(x3,(x1 * (x2 * ZERO)))>>
ANSWER PLEASE > < UNDEF or QUIT
<
<<((x1 * ZERO) + ZERO) = G(x2,((x1 * ZERO) + ZERO))>>
ANSWER PLEASE > < UNDEF or QUIT
<
<<(x1 + S(ZERO)) = S(x1)>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<G(x1,((x2 * S(x1)) + S(x1))) = (G(x1,(x2 * S(x1))) + G(x1,S(x1)))>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<G(x1,S(ZERO)) = F(x1)>>
ANSWER PLEASE > < UNDEF or QUIT
(* A direct definition of F(x) has been generated. The
transformation is terminated
but the completion will continue forever generating useless equations*)
QUIT
RESULT OF THE DELETION
[1  <<G(x1,ZERO)->ZERO>>;
2  <<(S(x1) * x2)->((x1 * x2) + x2)>>;
3  <<(ZERO * x1)->ZERO>>;
4  <<(S(x1) + x2)->S((x1 + x2))>>;
5  <<(ZERO + x1)->x1>>;
6  <<(x1 * S(ZERO))->x1>>;
7  <<G(ZERO,x1)->x1>>;
8  <<G(S(x2),x1)->G(x2,(x1 * S(x2)))>>;
9  <<(x1 + S(ZERO))->S(x1)>>;
10  <<F(x1)->G(x1,S(ZERO))>>]
Do you want to delete some rules YES or NO
YES
1 6 9
EQUATIONS
```

```
[]
RULES
[1  <<(S(x1) * x2)->((x1 * x2) + x2)>>;
2  <<(ZERO * x1)->ZERO>>;
3  <<(S(x1) + x2)->S((x1 + x2))>>;
4  <<(ZERO + x1)->x1>>;
5  <<G(ZERO,x1)->x1>>;
6  <<G(S(x2),x1)->G(x2,(x1 * S(x2)))>>;
7  <<F(x1)->G(x1,S(ZERO))>>]
```

9.6 Three transformation steps

```
GETTING THE SYSTEM
GETTING THE CONSTRUCTORS
C NIL
GROUND CONVERGENCE
WARNING
IF is not a constructor or a defined symbol
WARNING
ISSIG is not a constructor or a defined symbol
======================
Successful Completion
======================
RULES
[1  <<REPEAT(x1,C(x2,x3))->C(x1,REPEAT(R(x1),x3))>>;
2  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
3  <<SIGPERM(x1)->FILTER(CONCALL(x1))>>;
4  <<REPEAT(x1,NIL)->NIL>>;
5  <<FILTER(NIL)->NIL>>;
6  <<R(NIL)->NIL>>;
7  <<CONCALL(NIL)->NIL>>;
8  <<(NIL @ x1)->x1>>;
9  <<ALL(x1)->REPEAT(x1,x1)>>;
10  <<R(C(x2,x1))->(x1 @ C(x2,NIL))>>;
11  <<CONCALL(C(x1,x2))->(REPEAT(x1,x1) @ CONCALL(x2))>>;
12  <<FILTER(C(x1,x2))->IF(ISSIG(x1),C(x1,FILTER(x2)),FILTER(x2))>>]
The main symbol is: SIGPERM
SIGPERM TRANSFORMATION
Eureka_rules
[<<FILTER(CONCALL(x1))->CONCFIL(x1)>>]
Added Lemmas
[<<FILTER((x1 @ x2))->(FILTER(x1) @ FILTER(x2))>>]
you will be asked at each step if you want to add a lemma
ADD A LEMMA?? ANSWER YES or RETURN
<<CONCFIL(NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<(FILTER(REPEAT(x1,x1)) @ CONCFIL(x2)) = CONCFIL(C(x1,x2))>>
```

```
ANSWER PLEASE > < UNDEF or QUIT
<
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
=====================
Successful Completion
=====================
RESULT OF THE DELETION
[1  <<CONCFIL(NIL)->NIL>>;
2  <<FILTER(C(x1,x2))->IF(ISSIG(x1),C(x1,FILTER(x2)),FILTER(x2))>>;
3  <<R(C(x2,x1))->(x1 @ C(x2,NIL))>>;
4  <<(NIL @ x1)->x1>>;
5  <<R(NIL)->NIL>>;
6  <<FILTER(NIL)->NIL>>;
7  <<REPEAT(x1,NIL)->NIL>>;
8  <<SIGPERM(x1)->CONCFIL(x1)>>;
9  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
10  <<REPEAT(x1,C(x2,x3))->C(x1,REPEAT(R(x1),x3))>>;
11  <<CONCFIL(C(x1,x2))->(FILTER(REPEAT(x1,x1)) @ CONCFIL(x2))>>]
Another transformation step? YES or NO
YES
SIGPERM TRANSFORMATION
Eureka_rules
[<<FILTER(REPEAT(x1,x2))->SIGROT(x1,x2)>>]
Added Lemmas
[]
you will be asked at each step if you want to add a lemma
ADD A LEMMA?? ANSWER YES or RETURN
<<SIGROT(x1,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<SIGROT(x1,C(x2,x3))
 = IF(ISSIG(x1),C(x1,SIGROT(R(x1),x3)),SIGROT(R(x1),x3))>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
=====================
Successful Completion
=====================
RESULT OF THE DELETION
[1  <<SIGROT(x1,NIL)->NIL>>;
2  <<R(C(x2,x1))->(x1 @ C(x2,NIL))>>;
3  <<(NIL @ x1)->x1>>;
4  <<CONCFIL(NIL)->NIL>>;
5  <<R(NIL)->NIL>>;
6  <<SIGPERM(x1)->CONCFIL(x1)>>;
7  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
```

```
8  <<CONCFIL(C(x1,x2))->(SIGROT(x1,x1) @ CONCFIL(x2))>>;
9  <<SIGROT(x1,C(x2,x3))
   ->IF(ISSIG(x1),C(x1,SIGROT(R(x1),x3)),SIGROT(R(x1),x3))>>]
Another transformation step? YES or NO
YES
SIGPERM TRANSFORMATION
Eureka_rules
[<<(SIGROT(x1,x2) @ x3)->SR(x1,x2,x3)>>]
Added Lemmas
[<<(IF(x1,x2,x3) @ x4)->IF(x1,(x2 @ x4),(x3 @ x4))>>]
you will be asked at each step if you want to add a lemma
ADD A LEMMA?? ANSWER YES or RETURN
<<x1 = SR(x2,NIL,x1)>>
ANSWER PLEASE > < UNDEF or QUIT
<
<<SR(x1,C(x2,x3),x4)
 = IF(ISSIG(x1),C(x1,SR(R(x1),x3,x4)),SR(R(x1),x3,x4))>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
======================
Successful Completion
======================
RESULT OF THE DELETION
[1  <<SR(x2,NIL,x1)->x1>>;
2  <<(C(x1,x2) @ x3)->C(x1,(x2 @ x3))>>;
3  <<(NIL @ x1)->x1>>;
4  <<SIGPERM(x1)->CONCFIL(x1)>>;
5  <<CONCFIL(NIL)->NIL>>;
6  <<R(NIL)->NIL>>;
7  <<R(C(x2,x1))->(x1 @ C(x2,NIL))>>;
8  <<CONCFIL(C(x1,x2))->SR(x1,x1,CONCFIL(x2))>>;
9  <<SR(x1,C(x2,x3),x4)
   ->IF(ISSIG(x1),C(x1,SR(R(x1),x3,x4)),SR(R(x1),x3,x4))>>]
Another transformation step? YES or NO
NO
```

9.7 Associativity and Commutativity by UNFAIL ASTRE

For the system:

```
[1  <<SUMDOT(x1,x2,x4,x5,x3)->(DOT(x1,x2,x3) + DOT(x4,x5,x3))>>;
2  <<DOT(x1,x2,ZERO)->ZERO>>;
3  <<DOT(x1,x2,S(x3))->(DOT(x1,x2,x3) + (I(x1,S(x3)) * I(x2,S(x3))))>>]
```

The script of the transformation in the mode unfailing is:

```
SUMDOT TRANSFORMATION
```

```
Eureka_rules
[<<(DOT(x1,x2,x3) + DOT(x4,x5,x3))->SD(x1,x2,x4,x5,x3)>>]
Added Lemmas
[<<(x1 + x2)=(x2 + x1)>>]
[<<(ZERO + x1)->x1>>;
<<((x1 + x2) + x3)->(x1 + (x2 + x3))>>]
ADD A LEMMA?? ANSWER YES or RETURN
<<SD(x1,x2,x3,x4,ZERO) = ZERO>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
<<
(DOT(x1,x2,x3) + ((I(x1,S(x3)) * I(x2,S(x3))) + (DOT(x4,x5,x3)
+ (I(x4,S(x3)) * I(x5,S(x3)))))) = SD(x1,x2,x4,x5,S(x3))>>
ANSWER PLEASE > < UNDEF or QUIT
<
(* The unfailing completion generates 27 equations before finding
the simplification rules:*)
<<(DOT(x1,x2,x3) + (x4 + (DOT(x5,x6,x3) + x7)))
 = (SD(x5,x6,x1,x2,x3) + (x4 + x7))>>
ANSWER PLEASE > < UNDEF or QUIT
>
(*Now the display is:*)
EQUATIONS
[1  <<SD(x1,x2,x3,x4,x5)=SD(x3,x4,x1,x2,x5)>>;
2  <<(x1 + x2)=(x2 + x1)>>;
3  <<(DOT(x1,x2,x3) + SD(x4,x5,x6,x7,x3))
   =(SD(x1,x2,x4,x5,x3) + DOT(x6,x7,x3))>>;
4  <<(DOT(x1,x2,x3) + (SD(x4,x5,x6,x7,x3) + x8))
   =(SD(x1,x2,x4,x5,x3) + (DOT(x6,x7,x3) + x8))>>;
5  <<(SD(x1,x2,x3,x4,x5) + DOT(x6,x7,x5))
   =(DOT(x3,x4,x5) + SD(x6,x7,x1,x2,x5))>>;
6  <<(DOT(x1,x2,x3) + ((I(x1,S(x3)) * I(x2,S(x3)))
   + (x4 + (DOT(x5,x6,x3) + (I(x5,S(x3)) * I(x6,S(x3)))))))
   = (SD(x1,x2,x5,x6,x3) + ((I(x5,S(x3)) * I(x6,S(x3)))
   + ((I(x1,S(x3)) * I(x2,S(x3))) + x4)))>>]

RULES
[1  <<((x1 + x2) + x3)->(x1 + (x2 + x3))>>;
2  <<(ZERO + x1)->x1>>;
3  <<DOT(x1,x2,S(x3))->(DOT(x1,x2,x3) + (I(x1,S(x3)) * I(x2,S(x3))))>>;
4  <<DOT(x1,x2,ZERO)->ZERO>>;
5  <<SUMDOT(x1,x2,x4,x5,x3)->SD(x1,x2,x4,x5,x3)>>;
6  <<(DOT(x1,x2,x3) + DOT(x4,x5,x3))->SD(x1,x2,x4,x5,x3)>>;
7  <<SD(x1,x2,x3,x4,ZERO)->ZERO>>;
8  <<(DOT(x1,x2,x3) + (DOT(x4,x5,x3) + x6))->(SD(x1,x2,x4,x5,x3) + x6)>>;
9  <<(DOT(x1,x2,x3) + (x4 + DOT(x5,x6,x3)))->(SD(x5,x6,x1,x2,x3) + x4)>>;
10  <<SD(x1,x2,x4,x5,S(x3))
```

```
    -> (SD(x4,x5,x1,x2,x3) + ((I(x1,S(x3)) * I(x2,S(x3)))
       + (I(x4,S(x3)) * I(x5,S(x3))))))>>;
11  <<(x1 + ZERO)->x1>>;
12  <<(DOT(x1,x2,x3) + (x4 + (x5 + DOT(x6,x7,x3))))
    ->(SD(x6,x7,x1,x2,x3) + (x4 + x5))>>;
13  <<(DOT(x1,x2,x3) + (x4 + (DOT(x5,x6,x3) + x7)))
    ->(SD(x5,x6,x1,x2,x3) + (x4 + x7))>>]

(*SD is recursive the transformation can be stopped*)
QUIT
AUTOMATIQUE DELETION
[1  <<(ZERO + x1)->x1>>;
2  <<SUMDOT(x1,x2,x4,x5,x3)->SD(x1,x2,x4,x5,x3)>>;
3  <<SD(x1,x2,x3,x4,ZERO)->ZERO>>;
4  <<SD(x1,x2,x4,x5,S(x3))
   -> (SD(x4,x5,x1,x2,x3) + ((I(x1,S(x3)) * I(x2,S(x3)))
        + (I(x4,S(x3)) * I(x5,S(x3))))))>>;
5  <<(x1 + ZERO)->x1>>]
(*After deletion of the rules 1 and 5 by the user, the result is:*)
[1  <<SUMDOT(x1,x2,x4,x5,x3)->SD(x1,x2,x4,x5,x3)>>;
2  <<SD(x1,x2,x3,x4,ZERO)->ZERO>>;
3  <<SD(x1,x2,x4,x5,S(x3)) ->  (SD(x4,x5,x1,x2,x3)
        + ((I(x1,S(x3)) * I(x2,S(x3))) + (I(x4,S(x3)) * I(x5,S(x3))))))>>]
```

9.8 Transformation by AC ASTRE

The program of the example in Section 9.7 above is transformed by AC ASTRE:

```
<<DOT(x,y,S(n))=(DOT(x,y,n) + (I(x,S(n))*I(y,S(n))))>>
<<DOT(x,y,ZERO)=ZERO>>
<<SUMDOT(x,y,u,v,n)=(DOT(x,y,n)+DOT(u,v,n))>>
Mode Associatif-Commutatif
Give the associative and commutative operators
+
SUMDOT AC TRANSFORMATION
Eureka_Rules
(DOT(x1,x2,x3) + DOT(x4,x5,x3))=SD(x1,x2,x4,x5,x3)
Other_Lemmas
(x1 + ZERO)=x1
(DOT(x1,x2,x3) + DOT(x4,x5,x3)) = SD(x1,x2,x4,x5,x3)
ANSWER PLEASE > < =  UNDEF or QUIT
>
(x1 + ZERO)=x1
ANSWER PLEASE > < =  UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN or QUIT
ADD A LEMMA?? ANSWER YES or RETURN or QUIT
ADD A LEMMA?? ANSWER YES or RETURN or QUIT
ADD A LEMMA?? ANSWER YES or RETURN or QUIT
```

```
ADD A LEMMA?? ANSWER YES or RETURN or QUIT
SD(x1,x2,x3,x4,ZERO) = ZERO
ANSWER PLEASE > < =  UNDEF or QUIT
>
(((I(x1,S(x2)) * I(x3,S(x2))) + (I(x4,S(x2)) * I(x5,S(x2))))
     + SD(x4,x5,x1,x3,x2))
 = SD(x1,x3,x4,x5,S(x2))
ANSWER PLEASE > < =  UNDEF or QUIT
<
(((I(x1,S(x2)) * I(x3,S(x2))) + (I(x4,S(x2)) * I(x5,S(x2))))
    + SD(x4,x5,x1,x3,x2))
= (((I(x1,S(x2)) * I(x3,S(x2))) + (I(x4,S(x2)) * I(x5,S(x2))))
    + SD(x1,x3,x4,x5,x2))
ANSWER PLEASE > < =  UNDEF or QUIT
QUIT
RESULT OF THE DELETION
[1  <<SD(x1,x2,x3,x4,ZERO)->ZERO>>;
2  <<SD(x1,x3,x4,x5,S(x2)) ->  ((I(x1,S(x2)) * I(x3,S(x2)))
          + (I(x4,S(x2)) * I(x5,S(x2))) + SD(x4,x5,x1,x3,x2))>>;
3  <<SUMDOT(x1,x2,x3,x4,x5)->SD(x1,x2,x3,x4,x5)>>]
```

9.9 Third transformation step for the program Prefixes

```
[1  <<G(x1,NIL)->NIL>>;
2  <<PREFIXES(C(x1,x2))->C(NIL,PP(C(C(x1,NIL),G(x1,x2)),PREFIXES(x2)))>>;
3  <<PP(NIL,NIL)->NIL>>;
4  <<PREFIXES(NIL)->C(NIL,NIL)>>;
5  <<PP(C(x1,x3),C(x2,x4))->C(C(x1,x2),PP(x3,x4))>>;
6  <<G(x1,C(x2,x3))->C(C(x1,NIL),G(x1,x3))>>]

PREFIXES TRANSFORMATION
Eureka_rules
[<<PP(G(x1,x2),x3)->K(x1,x3)>>]
ADD A LEMMA?? ANSWER YES or RETURN
<<K(x1,x2) = PP(NIL,x2)>>
ANSWER PLEASE > < UNDEF or QUIT
<
ADD A LEMMA?? ANSWER YES or RETURN
<<K(x1,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
ADD A LEMMA?? ANSWER YES or RETURN
<<K(x1,x2) = PP(C(C(x1,NIL),G(x1,x3)),x2)>>
ANSWER PLEASE > < UNDEF or QUIT
<
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
ADD A LEMMA?? ANSWER YES or RETURN
```

```
<<K(x1,x2) = K(x3,x2)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<K(x2,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<K(x2,NIL) = K(x4,NIL)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<K(x4,NIL) = NIL>>
ANSWER PLEASE > < UNDEF or QUIT
>
<<K(x4,NIL) = K(x2,NIL)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<K(x1,x2) = K(x1,x2)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
<<K(x1,x2) = K(x1,x2)>>
ANSWER PLEASE > < UNDEF or QUIT
UNDEF
ADD A LEMMA?? ANSWER YES or RETURN
<<C(C(C(x1,NIL),x2),K(x1,x3)) = K(x1,C(x2,x3))>>
ANSWER PLEASE > < UNDEF or QUIT
<
ADD A LEMMA?? ANSWER YES or RETURN
<<PP(C(C(x1,NIL),NIL),x2) = K(x1,x2)>>
ANSWER PLEASE > < UNDEF or QUIT
QUIT
AUTOMATIQUE DELETION
[1  <<PREFIXES(NIL)->C(NIL,NIL)>>;
2  <<PREFIXES(C(x1,x2))->C(NIL,K(x1,PREFIXES(x2)))>>;
3  <<K(x1,NIL)->NIL>>;
4  <<K(x1,C(x2,x3))->C(C(C(x1,NIL),x2),K(x1,x3))>>]
```

References

[1] L. Bachmair. Proofs methods for equational theories. PhD thesis, University of Illinois, Urbana-Champaign, 1987. Revised version, August 1988.

[2] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In *Proceedings of the colloquium on the resolution of Equations in Algebraic Structures*, 1987.

[3] F. Bellegarde. Program Transformation and Rewriting. In *Proceedings of the fourth conference on Rewriting Techniques and Applications*, Springer Verlag, Lecture Notes in Computer Science 488, pages 226-239, Como, Italy, 1991.

[4] R. M. Burstall and J. Darlington. A Transformation System For Developing Recursive Programs. *Journal of the Association for Computing Machinery*, 24, pages 44-67, 1977.

[5] A. BenCherifa and P. Lescanne. Termination of Rewriting Systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9:137-160, 1987.

[6] N. Dershowitz. Termination. In *Proceedings of the first Conference on Rewriting Techniques and Applications*, Springer Verlag, Lecture Notes in Computer Science 202, pages 180-224, Dijon, France, 1985.

[7] N. Dershowitz. Synthesis By Completion. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 208-214, Los Angeles, 1985.

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

[9] N. Dershowitz. Completion and its Applications. *Resolution of Equations in Algebraic Structures,* Academic Press, New York, 1988.

[10] Projet FORMEL. *The CAML Reference Manual. Technical Report*, INRIA-ENS, March 1989.

[11] H. Ganzinger, A completion procedure for conditional equations, *Proceedings First International Workshop on Conditional Term Rewriting Systems*, S. Kaplan and J. P. Jouannaud, editors, Lecture Notes in Computer Science 308, pages 62-83, Orsay, France, 1987.

[12] G. Huet, Formal Structures for Computation and Deduction, Technical Report, INRIA, May 1986.

[13] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica,* 24:395-415, 1987.

[14] D. E. Knuth and P. B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract algebra*, pages 263-297, Pergamon Press, Oxford, U. K.,1970.

[15] E. Kounalis, Testing for Inductive (CO)-Reducibility. In *Proceedings of the 15th International Colloquium on Trees in Algebra and Programming,* Lecture Notes in Computer Science 431, pages 221-238, 1990.

[16] E. Kounalis, M. Rusinowitch. Mechanizing Inductive Reasoning. In *Proceedings of the eight National Conference on Artificial Intelligence*, AAAI-90, 1990.

[17] D.S. Lankford. On proving term rewriting systems are Noetherian, *Memo MTP-3,* Mathematic Department, Louisiana Tech. University, Ruston, LA, May 1979. (Revised October 1979).

[18] P. Lescanne, Completion Procedures as Transition Rules + Control,In *Proceedings of the Colloquium Current Concept In Programming Languages*, TAPSOFT, Springer Verlag, Lectures Notes in Computer Science 351, pages 28-41, Barcelone, Spain, 1989.

[19] P. Lescanne, Implementation of completion by Transition Rules + Control:ORME. In *2nd Intern. Workshop Algebraic and Logic Programming,* Springer Verlag, Lecture Notes in Computer Science 463, pages 262-269, 1990.

[20] G. E. Peterson and M. E. Stickel, Complete sets of reductions for some equational theories, *J. of the association for Computing Machinery*, 28(2), pages 233-264, 1981.

[21] U. S. Reddy. Transformational derivation of programs using the Focus system. In *Symposium Practical Software Development Environments*, pages 163-172, ACM, December 1988.

[22] U. S. Reddy, Formal methods in transformational derivation of programs. In Proceedings of the ACM Intern. Workshop on Automatic Software Design, AAAI, 1990.

[23] U. S. Reddy, Term Rewriting Induction. In *Proceedings of the Conference of Automated Deduction,* 1990.

[24] H. Zhang and J. L. Remy, Contextual rewriting. In *Proceedings of the First International Conference in Rewriting Techniques and Applications*, Springer Verlag, Lecture Notes in Computer Science 202, pages 46-62, Dijon, France, 1985.