THE REACHABILITY PROBLEM FOR GROUND TRS AND SOME EXTENSIONS

A. DERUYVER and R. GILLERON
LIFL UA 369 CNRS
Universite des sciences et techniques de LILLE FLANDRES ARTOIS
U.F.R. d' I.E.E.A. Bat. M3 59655 VILLENEUVE D'ASCQ CEDEX FRANCE

ABSTRACT

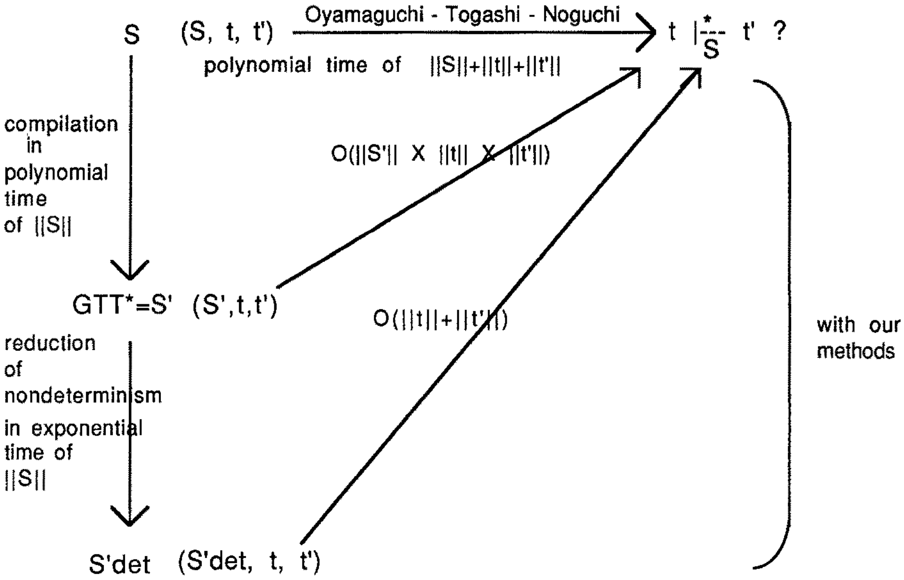
The reachability problem for term rewriting systems (TRS) is the problem of deciding, for a given TRS S and two terms M and N, whether M can reduce to N by applying the rules of S.
We show in this paper by some new methods based on algebraical tools of tree automata, the decidability of this problem for ground TRS's and, for every ground TRS S, we built a decision algorithm. In the order to obtain it, we compile the system S and the compiled algorithm works in a real time (as a fonction of the size of M and N).
We establish too some new results for ground TRS modulo different sets of equations : modulo commutativity of an operator σ, the reachability problem is shown decidable with technics of finite tree automata; modulo associativity, the problem is undecidable; modulo commutativity and associativity, it is decidable with complexity of reachability problem for vector addition systems.

INTRODUCTION

The reachability problem for term rewriting systems (TRS) is the problem of deciding, for given TRS S and two terms M and N, whether M can reduce to N by applying the rules of S. It is well-known that this problem is undecidable for general TRS's .In a first part we study this problem for more simple systems, more specifically in the case of ground term rewriting systems.
A TRS is said to be ground if its set of rewriting rules R={ li->ri | i∈I} (where I is finite) is such that li and ri are ground terms(no variable occurs in these terms). The decidability of the reachability problem for ground TRS was studied by Dauchet M. [4],[5] as a consequence of decidability of confluence for ground TRS. Oyamaguchi [15] and Togushi-Noguchi have shown this result too for ground TRS and in the same way for quasi-ground TRS.We take again this study with two innovator aspects:
- the modulary aspect of the decision algorithm which use all algebraical tools of tree automata, that permits to clearly describe it.
- the exchange between time and space aspect which have permitted to obtain some time complexities more and more reduced.
Therefore we have proceeded in three steps:
1- We begin with the TRS S not modified which gives the answer to the problem with a time complexity not bounded.

supported by "Greco programmation" and "PRC mathematique et informatique"

2- We transform the system S in a GTT (ground tree transducer) which simulates it, we will call this system, S'. Then the decision algorithm will have a quadratic time complexity. The memory space of S' will be in O((number of rules of S)^2).
3- Then, we obtain, after a compilation of S' which could be realised in an exponential time (reduction of nondeterminism), a real time decision algorithm (linear complexity). The necessary memory space, after the compilation of S', will be in O(exp(number of rules of S)).
If we make a comparison with the result of Oyamaguchi M.[15] we can have the next figure:



S = rewriting system
S'= our system S after compilation
S'det = our system S' after the reduction of nondeterminism
t,t' = the given trees
||S|| = size of the rewriting system S
||t|| = size of the tree t

A program, which is called VALERIANN, written in PROLOG realizes at the present time this algorithm.(on SUN machine)
In a second part, we consider the case of a ground TRS R_G modulo different sets of equations in the next three cases:

E_C :commutativity of an operator σ

E_A :associativity of an operator σ

E_{AC} :associativity and commutativity of an operator σ

R_C,R_A,R_{AC} denote the TRS obtained by orientation of equations into rules.We look at the two next problems:

For a TRS S equal to $R_G \cup R_C$,$R_G \cup R_A$,$R_G \cup R_{AC}$ and with conditions on the configuration of terms (i) if F is a recognizable forest, is the class of F modulo S recognizable ?(ii)decidability of the reachability problem
We have different results for each case:

For R_{GC}, we have a positive answer for (i) and henceforth for (ii)
For R_{GA}, we have a negative answer for the problems (i) and (ii)
For R_{GAC}, we have a negative answer for (i) and a positive answer for (ii) with the complexity of the reachability problem for vector addition systems.

I-PRELIMINARIES

Let us recall some classical definitions and some usefull results:
1- tree automata and recognizable forests.

Let Σ be a finite ranked alphabet.
T_Σ is the set of terms (or trees) over Σ.
Definition1: A frontier-to-root (bottom-up) tree automaton is a quadruplet $M=(\Sigma , Q , F, R)$ where

* Σ is a finite ranked alphabet.
* Q is a finite set of states.
* F is the set of final states, with F Q
* R is a finite set of transition rules, these rules have the next configuration: $c(q_{i1}[x1], \dots , q_{in}[xn])$ --> $q[c(x1, \dots ,xn)]$

if n=0, the rule is c' --> q[c']

We can dually define root-to-frontier(top-down) tree automata.
For more development see Gecseg F. & Steinby M.[7].
Definition2: A forest F is said to be recognizable if and only if there is a frontier-to-root tree automaton which accepts it.
properties: the class REC of recognizable forests is closed under union, intersection, and complementary.

2-algorithm of decision on tree automata
notation:
we note $||m||$ the number of rules of the automaton m and $|m_q|$ the number of states of the automaton m.
we note $\underline{m}$ the automaton which accepts the complementary of the language accepted by m

a-Decision of the emptiness ($M= \emptyset$)
Let M an automaton. The time complexity to answer to the next problem:
Is the language which is accepted by M empty ?
is:
* linear, for word languages, if we have direct access to rules and if we use a naive algorithm.
* in $O(||M||\times|M_q|)$, for tree languages.

b-Intersection of two automata M and M', and decision of the emptiness of this intersection. ($M \cap M' \neq \emptyset$)
* for word languages, the time complexity to answer to this problem is in $O(||M||\times||M'||)$.
* for tree languages, the time complexity is more important, it is in $O(||M||\times||M'||\times|M_q|\times|M_{q'}|)$.

c- Equivalence of M and M' (M=M')

$$M=M' \iff M \cap \underline{M}' = \varnothing \quad \text{and} \quad \underline{M} \cap M' = \varnothing$$

*deterministic case:
we can transform M in $\underline{M}$ by exchanging final states and the other states. Then we return to the same case than b-.

* nondeterministic case:
the time complexity contains the time of reduction of nondeterminism which is exponential.

3- Ground TRS and GTT.

*A tree rewriting system (TRS) S on T_Σ is a set of directed rewriting rules $R=\{li \dashrightarrow ri \mid i \in I\}$. Here, we only consider finite TRS (where I is finite), For more development see Huet G. & Oppen D.[8].
|--- is the extension of --> according to tree substitutions.
The reduction relation |-*- on T_Σ is the reflexive and transitive closure of -->.
S is a ground TRS if and only if no variable occurs in rules.

*A ground tree transducer on T_Σ (a GTT in short) is the relation T or (G,D) associated with two tree automata G and D and defined as follows:

$$t \overset{T}{\text{-><-}} t' \text{ iff there exist } u \in T_{\Sigma \cup Eg \cup Ed} \text{ such that } t \underset{G}{\to} u \underset{D}{\leftarrow} t'.$$

where Σ is a finite ranked alphabet.
Eg and Ed are sets of states.
In order to produce actual pairs of terms, the set Eg and Ed are supposed non disjoint. $Eg \cap Ed$ is called the interface.

*Dauchet M. and Tison S., and Dauchet , Heuillart, Lescanne and Tison have proved the next results:

<u>Proposition1:</u> There is an algorithm which associates to each ground TRS S a GTT Ts such that S = Ts where:

$$S = \{ (t,t') \mid t \underset{S}{\text{|-*-}} t' \} \quad \text{and} \quad Ts = \{(t,t') \mid t \underset{G}{\dashrightarrow} u \underset{D}{\leftarrow\!\!-\!\!-} t'\}.$$

<u>Proposition2:</u> The confluence of ground TRS is decidable.

<u>Proposition3:</u> The reachability problem for ground TRS is decidable.

<u>Proposition4:</u>
If F is recognizable then $[F]_S = \{ t' \mid \exists\, t \in F,\ t \underset{S}{\text{|-*-}} t' \}$ is recognizable.

II- COMPILATION OF A GROUND TRS S AND DECISION ALGORITHM FOR THE REACHABILITY PROBLEM

We will construct systems S' and S" , from the ground rewriting system S, so as to reduce more and more the time of answer to the reachability problem .
To do that, we use the next tools: Automata, Recognizable forest and ground tree transducer.

1-Creation of the system S'.

All along of the different steps, we will use the same example, so as to easily follow the different transformations which are realized.
Let us write the next ground rewriting system:

```
Σ ={ b1, q , q1' , p1 , b , q', p,   a   ,   c  }
                        |   |   |    /  \  /  |  \
```

rules = 1- b(b1) -> b1   2- a(b1, q) -> q   3- q1' -> q'(q1')

4- q(q1') -> q1'   5- q1' -> a (q1', q1')   6- b(q'(q'1)) -> c(p1, p(p1), p1)

7-p1 -> p(p1)   8- a(b1, a(q, b1)) -> b(q1')

First step:

In this part, we have to construct a GTT, from the system S, its frontier-to-root automaton will accept left hand sides of rules of S, and its root-to-frontier automaton will generate right hand sides of rules of S. Its interface states will make the connexion between left hand sides and right hand sides. for example we built for the rule 8 a frontier-to-root automaton which accepts the left hand side, where the terminal state is i8, and the other states are e14, e15, e16, e17.
Consider again our last system, then we will have the next rules:

	frontier-to-root automaton G	root-to-frontier automaton D
1-	b1 -> e1 b(e1) -> i1	i1-> b1
2-	b1-> e2 q-> e3 a(e2, e3) -> i2	i2 -> q
3-	q1'-> i3	i3 -> q'(e4) e4 -> q1'
4-	q1' -> e5 q'(e5) -> i4	i4 -> q1'
5-	q1' -> i5	i5 -> a(e6, e6) e6 -> q1'
6-	q1'-> e7 q'(e7) -> e8 b(e8) -> i6	i6 -> c(e9, e10, e11) e9 -> p1 e12 -> p1 e11 -> p1 e10 -> p(e12)
7-	p1 -> i7	i7 -> p(e13) e13 -> p1
8-	q -> e14 b1 -> e15 b1 -> e17 a(e14, e15) -> e16 a(e17, e16) -> i8	i8 -> b(e18) e18 -> q1'

Interface states are: I={ i1,i2,i3,i4,i5,i6,i7,i8}

Second step:
Creation of the GTT, G*, which simulates the ground rewriting system S.
The principle is:

" it's not good generating, to nible"

To do that , we create some ε-transitions, with the next induction rules:

$$\frac{\begin{array}{c} e \to f(e1, \ldots, en) \\ e1 \to e1', \ldots, en \to en' \\ f(e1', \ldots, en') \to e' \end{array}}{e \to e'}$$

The algorithm is :

1- we take a rule of the root-to-frontier automaton D

2- We examine, if we can find the right hand-side of this rule in the left hand-side of one rule of the frontier-to-root automaton G.

* if it is the case, we create an ε-transition with in left hand-side, the left hand side of the rule of D which is choosen (a state of D), and in right hand side, the state in which we arrive when we apply the rule of G which was found.Then we choose the next rule of D and we start again in 2.

* if it is not the case, we choose a new rule of D and we start again in 2.

Such a transformation can be illustrated with the diagram of the figure 2.This operation is realized in a polynomial time of n where n=||G||x||D|| .

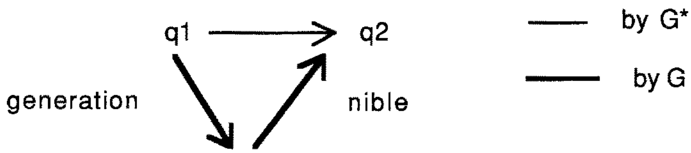


figure 2

Example:
Consider the rules 3 and 4 of the system S
The rule 3 was decomposed as follows: q1'->i3 i3-> q'(e4) e4->q1'
And the rule 4 was decomposed in this way: q1' ->e5 i4->q1' q'(e5) -> i4

Consider the state e4 ,we get: e4 -> q1' and q1' ->e5, q1' ->i3
so we get e4 -> e5 and e4 -> i3

Consider now the state i3 we get: i3 -> q'(e4)
and by the last step we get e4 ->e5 so i3-> q'(e5)
And we find q'(e5) -> i4 in the decomposition of the rule 4
So we deduce the next ε-transition i3 -> i4

So, instead of doing the next rewritings:

i3 -> q'(e4) -> q'(q1') -> q'(e5)-> i4

the GTT, G*, will directly pass by i3 to i4
So we have constructed in two steps, a GTT, denoted by G*, which simulates the system S, we call G* , the system S'. The answer to our problem will be given with S' in a quadratic time.

2-Creation of the system S".

In first time, we modify again the system S', so as to construct a frontier-to-root automaton. This one will accept a forest, which symbolizes all transformations that we can realize with the system S.
We can depict a tree which belongs to this forest like that:

```
                       O
                       |
                       a
                     /   \
                    b     c
                    .    / .
                   .    α   .
                   b          c
                  /         /   \
                 #         α     #
                / \       /       \
               b   b1    a         c
A=             .        / \       /|\
              .        b   a    p1 p p1
             b1        .  / \        .
                      .  b   a1      .
                     b1   .          p1
                          .
                          b1
```

Inside this tree, we can bring to light, two trees t and t', with two morphisms φ and φ' .

by φ, we get:

```
           O
           |
           a
          / \
         b   c
         .  / .
t=       . α   .
         b      c
         |     / \
         b    α   a
         .       / \
         .      b   a
         b1     .  / \
               .  b   a1
              b1   .
                   .
                   b1
```

φ erases the right son of each node #

by φ', we get:

```
            O
            |
            a
           / \
          b   c
          .  / .
t'=       . α    .
          b       c
          |      / \
          b1    α   c
                   /|\
                 p1 p p1
                      .
                      .
                      p1
```

φ' erases the left son of each node #

Like that, the tree A means that we can transform t in t' with the system S'.
So, all transformations according to the system S', are coded in a recognizable forest F.

with F={ t#t' | t $\overset{*}{\underset{S}{\vdash\!\!-}}$ t'}

To create a frontier-to-root automaton which will accept this forest, we proceed in three steps:

1- We keep nible rules of the sytem S'

2- We reverse generation rules of the system S' so as to convert them in bottom-up rules (by reversing the arrows)

3- For interface states we add next rules:

if i1=i2 with i1 a state of G and i2 a state of D

#(i1, i2) -> ok and #(i1, i2) -> (i1,i2)

and (i1,i2) -> ok

and then, for the other pairs of states, rules which have the next configuration:

if $e1 \neq e2$ with e1 a state of G and e2 a state of D

#(e1, e2) -> (e1,e2)

for all letters 'a' of Σ, we add, when it is possible, rules as follows:

a((e1,e1'), (e2,e2'), . . . , (en,en')) -> (e,e')

Finally, when we know that the 'ok' state allows to climb up to the root of the tree, we add, for all letter 'a' of the alphabet, rules as follows:

a(ok, ok, . . . , ok) -> ok

but, now, in order to improve the time complexity, we obtain the automaton S'', by transforming F and by reducing the nondeterminism.

1- Suppression of a hidden difficulty:

We bring down, into F, # nodes, the lower as possible, so that descendant letters of the # node would be always different.

ie: The next tree t:

```
t=    #        will be replace by       b
    /   \                             /   \
   b     b                           #     #
 /  \  /  \                         / \   / \
```

in order that our automaton could accept F with this modification. we must bring it some new changes:

a- We keep in states the last letter which is accepted.

b- We will not create rules as follows:

#(i1, i2) -> ok if i1 and i2 had accepted the same letter.

2- Reduction of nondeterminism.

We do that in an exponential time, in the worst cases.

When we have made all these different steps, we obtain S''.This one have a number of rules which is running to exp(number of rules of S) (not very readable), which are those of a frontier-to-root deterministic automaton S''.

The answer to the question: "can t be transformed into t'?" by the system S, is made in real time, because:

$t \mathrel{\vdash\!\!\overset{*}{-}}_{S} t' \quad \Leftrightarrow \quad$ t#t' is accepted by S''

Remark:
But the reduction of the nondeterminism stands some very important problems, all at once of memory space and of time of answer.
To avoid this problem, we can consider another method. We will see this method in the next paragraph.

3-Resolution of the reachability problem by using the system S' (G*)

Let us take G and D, which are automata of the GTT G* (see p6), and let us take Mt and Mt', which are automata which accept t and t'.
We call D_{inv}, the automaton obtained from D by reversing its arrows. So D_{inv} is a frontier-to-root automaton.
To solve our problem we can study two cases:

a- When G and D_{inv} are nondeterministic

We can answer to $t \underset{S}{\vdash\!\!-^*\!-} t'$? by using F

In fact $t \vdash\!\!-^*\!- t' \Leftrightarrow \varphi{-1}(t) \cap \varphi'{-1}(t') \cap F$

φ and φ' are morphisms which are defined above (p7). These one are independant of t, t' and S.
So we have a complexity equal to K(S) x || Mt || x || Mt'|| by omitting the access time.
Besides, we can proceed in the same way to express the set of all transformations of t, that we will call S(t), because:

$S(t)= \varphi'(\varphi{-1}(t) \cap F)$

The creation of the automaton which accepts S(t) is made with the next algorithm:
1-We make the intersection between the automaton Mt and the frontier-to-root automaton G of the GTT G*, but this thing by keeping all rules which accept t.
2-We search inside this automaton, rules which conduct to a couple of states (q,i) where i is an interface state of the GTT and q is any state, and we add all rules of the root-to-frontier automaton D of the GTT which start from this interface state i (this by reversing the arrows so as to always have a frontier-to-root automaton).
Such an algorithm is realized with a time complexity in $O((\|Mt\| \times \|G\| \times |Mt_q| \times |G_q|) + \|D\|)$.We will call this new automaton M_{st}
to answer to $t \vdash\!\!-^*\!- t'$, we make the intersection between the automata Mt' and M_{st}. So as to know if $S(t) \cap t' \neq \emptyset$
The answer is given after a time in $O(\|M_{st}\| \times \|Mt'\| \times |M_{stq}| \times |Mt'_q|)$

b- When G and D_{inv} are deterministic

As G is deterministic, it can accept the tree t, likewise for $D_{inv}$and t'.So we can, by recognition of t by G (resp of t' by D_{inv}), mark all subtrees of t which could be accepted by G (resp subtrees of t' accepted by D_{inv}).Our aim, is to have two new automata which will accept all at once t(or t') and trees which have the next configuration:

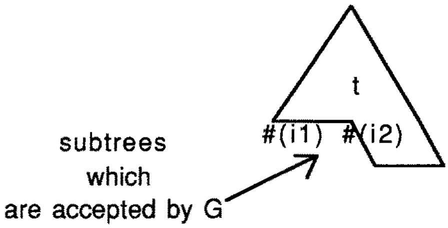


figure 3

where i1 and i2 are final states of the subtrees accepted by G (in fact, they are interface states), here, #(i1) and #(i2) replace these subtrees, they are leafs of the tree.

This operation is called, the "marking" operation.
Here,the algorithm used:

```
marking(x,l,y,e)
x: node
l: list of sons of the node x
y: state in which we arrive when we have accepted the node x (with rules of the automaton
Mt or Mt')
e: state in which we arrive when we have accepted the node x (with rules of the automaton G
or Dinv)
begin
if  it exists a rule of G (or Dinv) accepting the node x with the list l then
          - We keep the state e of G (or of Dinv) in which we arrive
              after having made the recognition.
           - We search if this state is an interface state :
          if yes then we add the next rule
                              #(e) -> state(y) in front of the list of rules of
                            the automaton Mt (or Mt').
                else nothing
          endif
else We keep a fictitious state 'p' so as to continue the
           exploration of the tree
             (remark: fathers of the node x ,couldn't be accepted by
          G(or Dinv))
endif
end

study-node(x,l,y,e)
begin
if the letter x is a leaf then  marking(x,l,y,e)
else if the letter x is a node then
                           for each son fi of x  do
                                    -Take the rules of Mt (or Mt') which conducts
                               to this son :
                              < xi,li>-> state(fi)
                                 -study-node(xi,li,fi,ei)
                                 -keep each ei in the list l'
                          end
                             marking(x,l',y,e)
     endif
endif
end
```

```
main program
begin
    Take the rule which accepts the root node of t or t'
     <x,l>->state(y)                x: root node
                                    l:list of sons
    study-node(x,l,y,e)
/ *exploration of the tree t (or t') with "marking" operation*/
end
```

The two automata obtained, after having applied this algorithm on t and t', are called Mt_m and Mt_m'.
we can remark that we make one and only one "marking" operation for each node of the considered tree (ie: for each rule of the associated automaton).
Besides, the "marking" operation of a node is made in a linear time, so we can deduce that the creation of automata Mt_m and Mt_m' is made with a time complexity in O(||Mt||+||Mt'||).
Now, we only have to compare these automata so as to find a tree common to the forest accepted by Mt_m and Mt_m',this tree will have the next configuration:

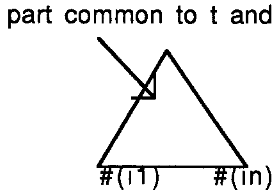


figure 4

i1, ..., in are interface states which represent all transformations made when we go from t to t'.
this operation is made in a very short time by using the next algorithm:

```
main program
 begin
     each automaton have only one final state, so we search the rules
     which conduct to these states:
   i.e.:     y-> state(ft)          ft and ft' are final states of Mtm and

             y1->state(ft')      Mtm'

      compare-node(y,y1,ft,ft') /* comparison of nodes y and y1*/
 end

compare-node(y,y1,e,e')
 begin
   if y=<x,L>  and y1=<x1,L1> then
        if x=x1 then consider each list
                    L=e1,e2,...,en and
                    L1= e1',e2',...,en'
                       (ei and ei' are states)
                     for each couple of states (ei,ei') do
                           -search rules which conduct to these two
                       states
                         i.e.: yi->state(ei)
                               yi'->state(ei')
                             -compare-node(yi,yi',ei,ei')
                    endfor
              else fail
        endif
```

```
else if y=#(e1) and y1=<x1,L1> then
                                 - take the rule so that y=<x,L>
                                      (this rule always exists)
                                 - compare-node(<x,L>,<x1,L1>,e,e')
    else if y=<x,L> and y1=#(e1') then
                                 -take the rule so that
                                 y1=<x1,L1>
                                      (this rule always exists)
                                 -compare-node(<x,L>,<x1,L1>,e,e')
    else if y=#(e1) and y1=#(e1) then OK
        else take rules so that y=<x,L> and y1=<x1,L1>
             compare-node(<x,L>,<x1,L1>,e,e')
        endif
      endif
     endif
   endif
end
```

We can see that in this case too, we only consider one and only one time, each rule of automata Mt_m and Mt_m'. So this algorithm is executed in a linear time, and the answer to our problem will be given after a time complexity in the order of $||Mt_m||+||Mt_m'||$.

We can compare this result with the result of complexity obtained in the paper of Oyamaguchi M.[15]. Their algorithm operates in a polynomial time of n where n= $||M_t||+||M_t'||+||S||$, where $||S||$ is the size of the given rewriting system. In our case, the complexity is began linear and the size of S is not consider. This fact can be explained because we have made a first operation of compilation on our rewriting system (this operation is made only once). So after this operation, we can ask as many questions as we want without making it again, that is why we earn much time.

Remark:
If G and D_{inv} are nondeterminist, the reduction of the nondeterminism on them is more realizable than on S" (the automaton which accepts all transformations that we can make with S), because, they have a smaller number of rules than S", so the time of execution of this operation is reduced.

III.Some extensions of ground TRS

Notation: Σ is a finite ranked alphabet

σ is a letter of arity 2 and $\sigma \notin \Sigma$

$\Delta=\Sigma\cup\{\sigma\}$

T_Σ, T_Δ are the set of terms (trees) over Σ,Δ

X is a set of variables

$T_\sigma(X)$ the set of terms over σ indexed by X

$T_\sigma(\Sigma)=\{t=t_\sigma(t_1,\ldots,t_n)/t_\sigma\in T_\sigma(X), \forall i, t_i\in T_\Sigma\}$

Let $R=\{l_i\rightarrow r_i/l_i,r_i\in T_\Sigma\}$ be a ground TRS on T_Σ and $R_\sigma=\{l_i\rightarrow r_i/l_i,r_i\in T_\sigma(\Sigma),l_i\notin T_\Sigma\}$ be a ground TRS on T_Δ, the condition $l_i \notin T_\Sigma$ is necessary because we consider terms in $T_\sigma(\Sigma)$ and recognizable forests included in $T_\sigma(\Sigma)$.

Let $R_G=R\cup R_\sigma$

Let $E_C=\{ \sigma(x, y) = \sigma(y, x)\}$ and $R_C=\{ \sigma(x, y) \rightarrow \sigma(y, x)\}$

Let $E_A=\{ \sigma(\sigma(x, y), z) = \sigma(x, \sigma(y, z))\}$

and $R_A=\{ \sigma(\sigma(x, y), z) \rightarrow \sigma(x, \sigma(y, z))\,;\, \sigma(x, \sigma(y, z)) \rightarrow \sigma(\sigma(x, y), z)\}$

and $E_{AC}=E_A\cup E_C$ and $R_{AC}=R_A\cup R_C$.

1.Commutativity

$R_{GC}=R_G \cup R_C = R \cup R_\sigma \cup R_C$ is the union of two ground TRS R and R_σ and of R_C TRS associated with commutativity of the operator σ

Example:

$R_{GC}=\{1{:}f(a)\to a\ ;2{:}\ g(a, a) \to b;3{:}\ \sigma(a, b) \to b;4{:}\ \sigma(x, y) \to \sigma(y, x)\ \}$

$R=\{1;2\}, R_\sigma=\{3\}, R_G=\{1;2;3\}, R_C=\{4\}$

1.1:Recognizability of $[F]_{R_{GC}}$

Lemma 1.1:There exists a TRS S_σ verifying: $\forall t,t' \in T_\sigma(\Sigma)$

$(t \underset{R_{GC}}{\mid\!-\!-*\!-\!-} t') \Leftrightarrow (\exists\ t_1,t_2 \in T_\sigma(\Sigma),\ t \underset{R}{\mid\!-*\!-} t_1 \underset{S_\sigma \cup R_C}{\mid\!-\!-\!-*\!-\!-\!-} t_2 \underset{R}{\mid\!-*\!-} t')$

Proof : - Construction of S_σ

We add to R_σ new rules to simulate rewritings by R on terms of T_Σ which appear in rules of R_σ.

$R_\sigma=\{l_i=l_{i\sigma}.(l_{i1},\dots,l_{ini})\to r_i=r_{i\sigma}.(r_{i1},\dots,r_{ipi})/i\in I, n_i>1; l_{i\sigma},r_{i\sigma}\in T_\sigma(X)\}$
$\forall i,j,k, l_{ij}, r_{ik} \in T_\Sigma\ \}$

Let $G=\bigcup_{i\in I}\{l_{i1},\dots,l_{ini}\}$ and $D=\bigcup_{i\in I}\{r_{i1},\dots,r_{ipi}\}$

Let $S=\{\ (r,l)\in DxG\ /\ r \underset{R}{\mid\!-*\!-} l\ \}$

DxG is finite and for every r of D the set $[r]_R=\{t'/\ r \underset{R}{\mid\!-*\!-} t'\}$ is recognizable ([4]) so we can construct S.

Let $R'=\{\ r\to l\ /\ (r,l) \in S\}$ and $S_\sigma=R_\sigma \cup R'$

- $\Leftrightarrow$

$\Leftarrow$ is obvious and $\Rightarrow$ is proved by induction on the number n of utilisations of rules of R_σ .It is based on the two results:

-Each rule of R can commute with each rule of R_C.

-Each rule of R' simulate the rewriting of a term of D in a term of G by R

So we obtain the decomposition of lemma 1.1, moreover we use a rule of R' only to transform a term of D in a term of G.

Lemma 1.2:There exists a TRS V_σ verifying: $\forall\ t,t' \in T_\sigma(\Sigma)$,

$(t \underset{R_{GC}}{\mid\!-\!-*\!-\!-} t') \Leftrightarrow (\exists t_1,t_2,t_3,t_4 \in T_\sigma(\Sigma)$,

$(\ t \underset{R}{\mid\!-*\!-} t_1 \underset{R_C}{\mid\!-*\!-} t_2 \underset{V_\sigma}{\mid\!-*\!-} t_3 \underset{R_C}{\mid\!-*\!-} t_4 \underset{R}{\mid\!-*\!-} t'\ )$

Proof : To obtain V_σ,we just add to S all the rules obtained by using commutativity of the operator σ on left-hand-side of rules of R_σ.

Proposition.1.3: F is a forest included in $T_\sigma(\Sigma)$

(F recognizable) $\Rightarrow$ ($[F]_{R_{GC}}$ recognizable)

Proof : $[F]_{R_{GC}}=\{t'\ /\ \exists\ t \in F\ ,\ t \underset{R_{GC}}{\mid\!-*\!-} t'\ \}$

For every recognizable forest F and every ground TRS S, the forest $[F]_S$ is recognizable([4]).For every recognizable forest F, the forest $[F]_{R_C}$ is recognizable (obvious with the bottom-up automaton recognizing F).So with the decomposition of lemma 1.2 we have $[F]_{R_{GC}}$ recognizable (R and V_σ are ground TRS).

1.2 :Reachability problem for R_{GC} in $T_\sigma(\Sigma)$

Proposition.1.4: For every t and t' in $T_\sigma(\Sigma)$, we can decide whether t can reduce to t' by applying rules of R_{GC} i.e the reachability problem is decidable for R_{GC} in $T_\sigma(\Sigma)$.

Proof : For every t in $T_\sigma(\Sigma)$,we have $[t]_{R_{GC}}$ recognizable (consequence of proposition 1.3) so we can decide if t' is in $[t]_{R_{GC}}$.

2.Associativity.

$R_{GA}=R_G\cup R_A=R\cup R_\sigma\cup R_A$ is the union of two ground TRS R and R_σ and of R_A TRS associated with associativity of the operator σ.

Example: $R_{GA}=\{1{:}\ f(a, a) \to a; 2{:}\ \sigma(\sigma(a, b), a) \to \sigma(b, b); 3{:}\ \sigma(\sigma(x, y), z) \to \sigma(x, \sigma(y, z)); 4{:}\ \sigma(x, \sigma(y, z)) \to \sigma(\sigma(x, y), z)\}$
$R=\{1\}$; $R_\sigma=\{2\}$; $R_G=\{1;2\}$; $R_A=\{3;4\}$

2.1:Recognizability of $[F]_{R_{GA}}$

Example: Let $R=R_\sigma=\emptyset$ and so $R_{GA}=R_A$ and F be the recognizable forest generated by the regular grammar $\{ A \to \sigma(a, \sigma(A, b)) \ ; A \to \sigma(a, b) \}$
Then $[F]_{R_{GA}}=[F]_{R_A}=\{ t\in T\{\sigma;a;b\}/\phi(t)=a^nb^n, n>0 \}$ where $\phi(t)$ denotes the frontier of the term t and $[F]_{R_{GA}}$ is not recognizable so in general F recognizable does not imply $[F]_{R_{GA}}$ recognizable

2.2:Reachability problem for R_{GA} in $T_\sigma(\Sigma)$

Proposition 2.1: The reachability problem for R_{GA} in $T_\sigma(\Sigma)$ is undecidable

Proof: Let Γ be a finite alphabet and R_w be a word rewriting system on Γ^* ,let $\Delta=\Gamma\cup\{\sigma\}$ be a finite ranked alphabet (all letters of Δ are of arity 0 except σ which arity is 2).

Let $f : \Gamma^* \to T_\Delta$

$m \mapsto f(m)$ defined by (if $|m|=1$ then $f(m)=m$)

and(if $|m|>1$ and $m=a_1a_2...a_n$ then $f(m)=\sigma(a_1, \sigma(a_2 ,\sigma (a_3, \dots \sigma(a_{n-1}, a_n))))$)

So we can associate to $R_w=\{ l \to r / l,r \in \Gamma^* \}$ a TRS denoted R_G defined by $R_G=\{ f(l) \to f(r)/ l\to r \in R_w \}$and thus we can prove $(t \overset{*}{\underset{R_{GA}}{\vdash}} t') \Leftrightarrow (\Phi(t) \overset{*}{\underset{R_w}{\vdash}} \Phi(t'))$

The reachability problem for R_w in Γ^* is known undecidable so the reachability problem for R_{GA} in $T_\sigma(\Sigma)$ is undecidable.

3.Associativity and commutativity.

$R_{GAC}=R_G\cup R_{AC}=(R\cup R_\sigma)\cup(R_A\cup R_C)$ is the union of the ground TRS R_G and of R_{AC} TRS associated with commutativity and associativity of the operator σ .R_G is itself the union of the ground TRS R on T_Σ and of the TRS R_σ on T_Δ(with $\Delta=\Sigma\cup\{\sigma\}$ and conditions on the configuration of rules of R_σ,see III Notations).

3.1: Recognizability of $[F]_{R_{GAC}}$

Example: With $R=R_\sigma=\emptyset$ and so $R_{GAC}=R_{AC}$ with the forest F of the example of the section III.2.1 we have $[F]_{R_{GAC}}=[F]_{R_{AC}}=\{ t \in T\{\sigma,a,b\}/ |\Phi(t)|_a=|\Phi(t)|_b \}$ (where $\Phi(t)$ is the frontier of the term t and $|\Phi(t)|_a$ the number of occurences of a in the word $\Phi(t)$) which is not recognizable .So generally F recognizable does not imply $[F]_{R_{GAC}}$ recognizable .

3.2: Reachability problem for R_{GAC} in $T_\sigma(\Sigma)$

Lemma 3.1: There exists a TRS S_σ such that : $\forall t,t' \in T_\sigma(\Sigma)$
$(t \overset{*}{\underset{R_{GAC}}{\vdash}} t')\Leftrightarrow(\exists t_1,t_2\in T_\sigma(\Sigma) , t \overset{*}{\underset{R}{\vdash}} t_1 \overset{*}{\underset{S_\sigma\cup R_{AC}}{\vdash}} t_2 \overset{*}{\underset{R}{\vdash}} t')$

Proof: Similar to lemma 1.1.

With the notations of section III.1.1,let $M=G\cup D=\{u_1,\dots,u_m\}$ be the set of all subterms of T_Σ which appear as subterms of rules of S_σ.

Example: $S_\sigma=\{ \sigma(f(a, a), c) \to \sigma(f(a, a), d) ;2{:}\ \sigma(a, f(a, a)) \to c ;3{:}\ \sigma(c, d) \to c \}$
then $M=\{ a , f(a, a) , c , d \}$

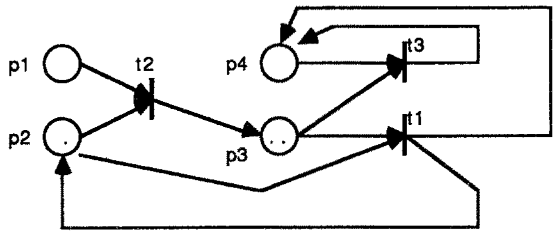
Let $X=\{x_1,...,x_m\}$ be an alphabet one to one with M
on X^* we define the relation $(m \equiv m') \Leftrightarrow (\forall x \in X, |m|_x = |m'|_x)$
Let $f : T_\sigma(\Sigma) \rightarrow X^*/_\equiv$
$t=t_\sigma.(t_1,...,t_n) \mapsto f(t)=x_1^{y1}...x_m^{ym}$ where yi is the number of occurences of the term u_i (which belongs to M) in $\{t_1,...,t_n\}$
Thus to each tree t of $T_\sigma(\Sigma)$, we can associate f(t) in $X^*/_\equiv$ and g(t) the list (or multiset) of terms of $\{t_1,...,t_n\}$ which are not in M (g(t) is the list of subterms of t which cannot be transformed by S_σ).
Example: $S_\sigma=\{1,2,3\}$;M={a, f(a, a) ,c,d};X={x,y,z,t}
with t= $\sigma(\sigma(\sigma(f(a, a), c), b), \sigma(c, b))$
We get $f(t)=yz^2$ and g(t)=(b,b)
Moreover to each rule $l_i \rightarrow r_i$ of S_σ, we can associate the rule $f(l_i) \rightarrow f(r_i)$ on $X^*/_\equiv$ and thus to S_σ is associated a TRS S_X on $X^*/_\equiv$.
Example: With S_σ,M,X defined in the previous example
we get $S_X=\{ 1_X : yz \rightarrow yt ; 2_X : xy \rightarrow z ; 3_X : zt \rightarrow z \}$

Lemma 3.2: $\forall\ t_1,t_2 \in T_\sigma(\Sigma)$
$(t_1 \xrightarrow[S_\sigma \cup R_{AC}]{*} t_2) \Leftrightarrow (f(t_1) \xrightarrow[S_X]{*} f(t_2)$ and $g(t_1)$ and $g(t_2)$ contain exactly the same terms)
Proof: -S_X is a TRS on $X^*/_\equiv$ and by definition of $X^*/_\equiv$ the rewritings are made modulo commutativity and associativity so each rule of S_X simulates commutativity,associativity and one rule of S_σ

-the trees of $g(t_1)$ cannot be rewritten by S_σ so we must have the second condition.

Example: With S_σ,M,X,t of the previous example we have
$t \xrightarrow[\{1\}]{} \sigma(\sigma(\sigma(f(a, a), d), b), \sigma(c, b)) \xrightarrow[R_{AC} \cup \{3\}]{*} t'= \sigma(\sigma(f(a, a), c), \sigma(b, b))$
and $f(t)=yz^2$, $f(t')=yz$, $yz^2 \xrightarrow[1_X]{} yzt \xrightarrow[3_X]{} yz$
g(t)=g(t')=(b,b).
Lemma 3.3: The reachability problem for S_X in $X^*/_\equiv$ is decidable.
Proof: To the TRS S_X on $X^*/_\equiv$,we can associate the Petri net PS_X defined as follow:
-Set of places $P=\{p_1,...,p_m\}$,p_i is associated with x_i of X
-Set of transitions $T=\{t_1,...,t_n\}$,t_i is associated with the rule $l_i \rightarrow r_i$.
-Pré and Post are defined by :
if $x_1^{li1}...x_m^{lim} \rightarrow x_1^{ri1}...x_m^{rim}$ is the rule $l_i \rightarrow r_i$ of S_X then for the transition t_i we have $Pré(p_j,t_i)=l_{ij}$ and $Post(p_j,t_i)=r_{ij}$
Moreover to each $m=x_1^{y1}...x_m^{ym}$ of $X^*/_\equiv$ we associate the vector v(m) of N^m such that $v(m)(i)=y_i$.
We can dually associate to a Petri net a TRS on $P^*/_\equiv$ so the reachability problem for S_X in $X^*/_\equiv$ is equivalent to the reachability problem for Petri net indeed decide if m can reduce to m' by applying rules of S_X is decide if the vector v(m') is reachable for the Petri net SP_X with the initial marking v(m).The reachability problem in Petri nets is decidable (Kosaraju[11],Mayr[13]) and so the reachability problem for S_X in $X^*/_\equiv$ is decidable.
Example: With S_σ,M,X={x,y,z,t},$S_X=\{1_X:yz \rightarrow yt;2_X:xy \rightarrow z;3_X:zt \rightarrow z\}$ and t of the previous example we have P={p,q,r,s} , T={t ,t',t"} and

$$Pre=\begin{pmatrix} 010 \\ 110 \\ 101 \\ 001 \end{pmatrix};Post=\begin{pmatrix} 000 \\ 100 \\ 010 \\ 101 \end{pmatrix};\text{Initial marking } v(m)=\begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \end{pmatrix}$$



Petri net SPX with the initial marking v(m)

Proposition 3.4: The reachability problem for R_{GAC} in $T_\sigma(\Sigma)$.is decidable
Proof:-If t and t' belong to T_Σ then R_{GAC}=R and the reachability problem for the ground TRS R is decidable

-If t belongs to T_Σ and t' does not belong to T_Σ,t cannot be rewrited in t' because of the condition $l_i \notin T_\Sigma$ for rules of R_σ(we forbid the generation of σ from terms of T_Σ).

-If $t=t_\sigma.(t_1,...,t_n) \in T_\sigma(\Sigma)$ and $t'=t'_\sigma.(t'_1,...,t'_p) \in T_\sigma(\Sigma)$ .We use the decomposition of lemma 3.1 so we first rewrite the terms t_i by the ground TRS R on T_Σ and so each term t_i can produce terms of $M=G \cup D$ or not so we consider $F(t)=\{t_\sigma.(u_1,...,u_n)/$ if $[t_i]_R \cap M=\emptyset$ then $u_i=t_i$;if $[t_i]_R \cap M=\{m_1,...,m_j\}$ then $u_i=t_i$ or $u_i=m_1$ or ... or $u_i=m_j$ }.M is a finite set of terms and for each t_i,the forest $[t_i]_R$ is recognizable so we can build the finite set F(t) for every t of $T_\sigma(\Sigma)$.Dually, using decomposition of lemma 3.1,we consider
$F^{-1}(t')=\{t'_\sigma.(u'_1,...,u'_p)$ / if $[t'_i]^{-1}{}_R \cap M = \emptyset$ then $u'_i=t'_i$;if $[t'_i]^{-1}{}_R \cap M=\{m'_1,...,m'_k\}$ then $u'_i=t'_i$ or $u'_i=m'_1$ or...or $u'_i=m'_k\}$ with $[t']^{-1}{}_R=\{t/\ t \vdash\!\!*\!\!-\ t'$ for the TRS R} which is recognizable so we can build the finite set $F^{-1}(t')$ for every t' of $T_\sigma(\Sigma)$.We are now ready to show

Lemma3.5:($t \ \underset{R_{GAC}}{\vdash\!\!-\!*\!-\!-}\ t'$)$\Leftrightarrow$(($\exists T \in F(t)$, $\exists T' \in F^{-1}(t')$, $f(T) \underset{S_X}{\vdash\!\!*\!-\!-} f(T')$) and (there exists a one to one correspondance h between g(T) and g(T') such that we have $g(T) \ni u \underset{R}{\vdash\!\!*\!-} h(u) \in g(T')$))

Proof: We use in this proof the results of lemma 3.1, lemma 3.2 and the construction of the finite sets F(t) and $F^{-1}(t')$.$\Leftarrow$ is without difficulty using these results.For $\Rightarrow$ we have to examine the rewriting of t in t' by R_{GAC} using the decomposition of lemma 3.1 and build T of F(t) and T' of $F^{-1}(t')$ verifying the two properties.

$$t= t_\sigma.(t_1,...,t_n) \underset{R}{\vdash\!\!*\!-} t_\sigma.(v_1,...,v_n)$$

$$t_\sigma.(v_1,...,v_n) \underset{S_\sigma \cup R_{AC}}{\vdash\!\!-\!-\!-\!*\!-\!-\!-\!-} t'_\sigma.(v'_1,...,v'_p)$$

$$t'_\sigma.(v'_1,...,v'_p) \underset{R}{\vdash\!\!*\!-} t'_\sigma.(t'_1,...,t'_p)=t'$$

This construction is not difficult,we just have to look at every possible cases for the rewritings by R: $t_i \vdash\!*\!- v_i \in M$; $t_i \vdash\!*\!- v_i \notin M$ and then $t_i \in M$ or $t_i \notin M$;and dually $M \ni v'_i \vdash\!*\!- t'_i$; $v'_i \notin M, v'_i \vdash\!*\!- t'_i$ and then $t'_i \in M$ or $t'_i \notin M$.

Moreover, F(t) and $F^{-1}(t')$ are finite sets, for every (t,t') of $F(t) \times F^{-1}(t')$, we can decide if the properties of lemma 3.5 are satisfied or not (lemma 3.4 and decidability of the reachability problem for the ground TRS R) and so the reachability problem for R_{GAC} in $T_\sigma(\Sigma)$ is decidable.

CONCLUSION

These works could permit to obtain some algebraical methods to realise the compilation of TRS, so as to have an execution of these sorts of systems in a real time.
Besides, these researches show the difficulty to have some good classes and make us researching some partial algorithms of decision of the reachability problem based on our methods for these classes.

BIBLIOGRAPHY

[1] BRAINERS : Tree-generating regular systems, Info and control (1969)

[2].G.W.BRAMS : Reseaux de Petri:theorie et pratique,tomes 1&2, Masson,Paris (1983)

[3] CHEW : An improved algorithm for computing with equations,21st FOCS (1980)

[4] DAUCHET, HEUILLARD, LESCANNE, TISON : The confluence of ground term tewriting systems is decidable,2nd LICS (1987)

[5] DAUCHET & TISON : Tree automata and decidability in ground term rewriting systems FCT' 85 (LNCS n° 199)

[6] N.DERSHOWITZ,J.HSIANG,N.JOSEPHSON and D.PLAISTED : Associative-commutative rewriting,Proc.10th IJCAI, LNCS 202 (1983)

[7] GECSEG F. & STEINBY M : tree automata,Akadémiai Kiado, Budapest (1984)

[8] HUET.G & OPPEN.D.: Equations and rewrite rules:a survey,in formal languages :perspective and open problems,Ed.Book R.,Academic Press(1980)

[9] J.P.JOUANNAUD : Church-Rosser computations with equational term rewriting systems,Proc.4th Conf on Automata, Algebra and programming,LNCS 159 (1983)

[10] C.KIRCHNER : Methodes et outils de conception systematique d'algorithmes d'unification dans les théories équationnelles,These d'etat de l'universite de Nancy I (1985)

[11] S.R.KOSARAJU : Decidability of reachability in vector addition systems, Proc.14th Ann.Symp.on Theory of Computing, 267-281.(1982)

[12] KOZEN : Complexity of finitely presented algebra, 9th ACM th. comp. (1977)

[13] E.W.MAYR : An algorithm for the general Petri net reachability problem, Siam J Comput.13 441.- 460

[14] NELSON & OPPEN : Fast decision algorithms based on congruence closure, JACM 27 (1980)

[15] OYAMAGUCHI M.: The reachability problem for quasi-ground Term Rewriting Systems, Journal of Information Processing , vol 9 , n°4 (1986)

[16] PLAISTED D. & BACHMAIR L. : Associative path ordering, Proc. 1st conference on Rewriting Techniques and Applications, LNCS 202 (1985)

[17] RAOULT J.C. : Finiteness results on rewriting systems, RAIRO, IT, vol 15 (1985)