DESCENDANTS OF REGULAR LANGUAGE IN A CLASS OF REWRITING SYSTEMS : ALGORITHM AND COMPLEXITY OF AN AUTOMATA CONSTRUCTION .

M.Benois
L.S.D. , I.M.A.G. , Grenoble 1 University ,
B.P.68 , 38402 St Martin d'Hères Cedex

Introduction :

Recent works on public key encryption for secure network communication [7] have brought back the following problem : given a regular set R on A^* ,defined by a non deterministic finite automaton with n states and a rewriting system T , how can we construct an automaton that recognizes the set of descendants of R : $\Delta^*(R)$ when this language is regular [1]. Some algorithms are found by Book and Otto [6] or Sakarovitch and me [3],in very particular cases of systems and gave complexity in $O(n^4)$ in [6] and $O(n^3)$ in [3] . Here we give a strong extension of these algorithms in a large class of systems however the complexity of our algorithm does not depend on the lenght of the words of T and is at most in $O(n^6)$.

1. Semi Thue systems : definitions and results .

Given a finite subset T of A^*xA^* , we can consider it either as a non symetric system and its elements (f,g) as rewriting rules , or as a symetric system which generates a congruence and which may be Church Rosser . In this work we study systems as rewriting rules and only in the conclusion we transform the results in the Church Rosser view point . Below some definitions and properties of the studied systems are given .

Let A be a finite set , A^* is the *free monoid* generated by A with the empty word 1 as identity . The *lenght* of a word f of A^* is denoted by |f| . A *Thue system* T is a finite subset of A^*xA^* ; T defines a regular relation denoted by $\rightarrow_T$, [9] and defined by :

$u \rightarrow_T v$ iff there exist x and y in A^* and (f,g) in T such that $u = xfy$ and $v = xgy$.

The transitive (resp. transitive and reflexive) closure of $\rightarrow_T$ is denoted by $\rightarrow^+_T$ (resp. $\rightarrow^*_T$). A word v is a *descendant* of a word u in T when

$u \rightarrow^{+}_{T} v$ and $\Delta^{+}(u)$ is the set of all the descendants of u and $\Delta^{*}(u) = \Delta^{+}(u) \cup \{u\}$. A *T-chain* $u_1, u_2, \ldots u_n$ is a sequence of words such that $\forall i \; u_i \rightarrow_T u_{i+1}$.

Let's recall some well known definitions:

-a word is *irreducible* for T if there exists no T-chain beginning with it .
-T is *noetherian* if every T-chain is finite . This implies that $\forall$ u in A^* , $\Delta^*(u)$ is finite ,[9].

and give some new ones :

-Two systems T and T' are *equivalent* if the their generated relations $\rightarrow^*_T$ and $\rightarrow^*_{T'}$ are equal . Then the sets of descendants of a word u are equal .
-Two systems T and T' are *confluently equivalent* if : whenever $u \rightarrow^*_T u'$ (resp. $u \rightarrow^*_{T'} u'$) then $\exists$ v such that $u \rightarrow^*_{T'} v$ and $u' \rightarrow^*_{T'} v$ (resp. $u \rightarrow^*_T v$ and $u' \rightarrow^*_T v$).This relation between systems is an equivalence when the systems are confluent .

It is clear that two equivalent systems are confluently equivalent and that the contrary is false . The decidability of the equivalence problem is implied by the decidability of the following particular word problems : given two words u and v , is one of them , a descendant of the other ,for T and T'.

Proposition 1.1 : *If T and T' are two equivalent systems ,they are noetherian at the same time .*

Now let us consider the properties of words which describe a system T and denote by :

- $H = \{ f \in A^* , \exists g \in A^* \; g \neq 1 \text{ and } (f,g) \in T \}$
- $N = \{ f \in A^* , \exists g \neq 1 \text{ and } (f,g) \in T \}$
- $C = \{ g \in A^* , g \neq 1 \text{ and } (f,g) \in T \}$

in certain cases may be $H \cup N \neq \emptyset$.

The properties of our systems will be defined below :

-T is *basic* if words in C do not overlap properly words in HUN ,that is :
$\forall f \in HUN , \forall g \in C$ if $\exists$ u,v,w,such that either f=uv and g=vw then w=1

or f=vu and g=wv then w=1 ;[12] and [13] .

-T is *semi-reduced* if a factor of a word in C is never in HUN .Or every word in C is irreducible ;[11].

-T is *reducible* if T is equivalent to a semi-reduced system .

-T is *confluently reducible* if T is confluently equivalent to a semi-reduced system .

The following propositions explain the aim of these definitions:

Proposition 1.2 : *A basic and semi-reduced system is noetherian* .

The proof goes by induction on the difference between the lenght of a word and the lenght of its factors in HUN . A useful consequence of proposition 1.2 is that in semi-reduced basic system $\forall u \in A^*$, $\Delta^*(u)$ is finite

Proposition 1.2 : *One can decide if a given system T is confluently reducible* .

This implies that if T is noetherian then T is confluently reducible .

2. Properties of sets related to a system T on A .

We omit the T in this part since T is always the same .Notations:

-$\Delta^1(u) = \{ v \in A^*$ such that $u \rightarrow v \}$.
-$\Delta^i(u) = \{ v \in A^*$ such that $\exists w \in \Delta^{i-1}(u)$ and $w \rightarrow v \}$.
-$\Delta_\#(u) = \{ v \in A^*$ such that $\exists$ n and $(f_1,g_1),(f_2,g_2),\ldots(f_n,g_n) \in T$ and $x_1,x_2,\ldots x_{n+1} \in A^*$ and $u = x_1f_1x_2f_2 \ldots f_nx_{n+1}$, $v = x_1g_1x_2g_2 \ldots g_nx_{n+1} \}$.

These definitions are extented to a set of words instead of a word and:

-$\Delta^i(P) = \cup_{u \in P} \Delta^i(u)$
-$\Delta^+(P) = \cup_{i \geq 1} \Delta^i(P)$, $\Delta^*(P) = \cup_{i \geq 0} \Delta^i(P)$.

We have the following properties :

-If $P \subset P'$ then $\Delta(P) \subset \Delta(P')$, $\Delta_{\#}(P) \subset \Delta_{\#}(P')$, $\Delta^*(P) \subset \Delta^*(P')$;

- $P \subset \Delta^*(P)$, $\Delta^+(\Delta^+(P)) = \Delta^+(P)$, $\Delta^*(\Delta^+(P)) = \Delta^+(P)$;

- $\Delta(P) \subset \Delta_{\#}(P) \subset \bigcup_{i \leq \max|u|, u \in P} \Delta^i(P)$;

- $\Delta_{\#}(P) = \Delta^+(P)$, $\Delta^*_{\#}(P) = \Delta^*(P)$;

-These operators are stable for union and intersection of subsets .

3. Descendants of regular languages in a semi-reduced basic system T .

The set of semi-reduced basic systems is a large extension of the well known and studied following systems:

-special systems where $H = \emptyset$

-monadic systems where $C \subset A$.

They are not all lenght-reducing , however when they are not ,$\Delta^*(u)$ remains finite ,by proposition 1.2 .

We refer to Hopcropft and Ullman [10] for the definitions of finite automata and of regular languages on A and we follow their notations . Given a regular language R on A accepted by a non deterministic finite state automaton $\mathcal{A} = (Q , A, q_0 , \partial , F)$,we want to transform this automaton in another that recognizes all the descendants of R : $\Delta^*(R)$ with respect to a semi-reduced , basic system T .The algorithm we describe and prove below is also proof of :

Theorem 1 :*The set* $\Delta^*(R)$ *of the descendants in a semi-reduced basic system of a regular language* R *is a regular language on* A .

The study of the complexity of the algorithm gives the next theorem :

Theorem 2 : *Let* T *be a finite semi-reduced basic system on* A *and* R *be a regular language on* A *specified by a non deterministic finite automaton with* n *states , one can effectively construct in* $O(n^6)$ *steps a non deterministic finite automaton that recognizes* $\Delta^*_T(R)$.

This theorem generalizes the algorithm described in [3] and its complexity in some particular cases is :

-$O(n^4)$ if $H = \emptyset$.

-$O(n^4)$ if T is monadic (i.e. for every $g \in C$ $|g| \leq 1$) .

-$O(n^4)$ if for every $f \in H \cup N$ $|f| \leq 2$.

-$O(n^3)$ if for every $f \in HUN$ $|f| \leq 2$ and for every $g \in C$ $|g| \leq 1$.

4. The basic principles of computing an automaton that recognizes $\Delta^*(R)$.

First we will consider the given automaton $\mathcal{A} = (Q, A, q_0, \partial, F)$ as a directed labelled graph $G(\mathcal{A})$ the vertices of which are elements of Q . Its edges represent the transition function ∂ : (q,x,q') is an edge from q to q' with label $x \in A$ iff $q' \in \partial(q,x)$; a vertex q_0 is the initial state and F is a subset of vertices . A path of $G(\mathcal{A})$ is a sequence of vertices and edges denoted by (q , u , q') where q is the beginning , q' the end and u is the concatenation of the labels of the edges . A word u is recognized by $\mathcal{A}$ iff a path (q_0,u,q') exists in $G(\mathcal{A})$ with $q' \in F$.

The main idea is : for every $f \in HUN$ and every path (q,f,q') in $G(\mathcal{A})$ we have to add a new path (q,g,q') for every g such that $(f,g) \in T$. This implies that we add new vertices and new edges and so create new paths (q,f,q') and the process may be infinite .We choose a particular way to add these new vertices and edges ; it is easy to prove that in this way the process is finite but we have then to prove that the graph we construct recognizes $\Delta^*(R)$.

Initially G_0 is the graph of $\mathcal{A}$ that recognizes R ,its vertices will be called *initial vertices* . The finiteness of the algorithm comes from the consideration of a queue ARC which , at the beginning , countains all the edges of G_0 labelled by a letter of an $f \in FUN$; each step of the algorithm can put new elements is this queue .

For every edge (i,x,j) at the front of the queue and every $f \in HUN$ we look, in the graph constructed at this step , for the paths (q,f,q') which contain the edge(i,x,j) ,this is *part one* in the algorithm .Whenever we found such a path we have two ways in adding new paths :

-if $f \in H$ we add, first new vertices and new edges related to these vertices to realise a path (q,g,q') for every g such that $(f,g) \in T$; then the first new edge created in the queue ARC. This is *part two* of the algorithm.

-if $f \in N$ we add, first a new edge (i,y,q") every time (j,y,q") is already an edge in the graph ,then this new edge in the queue ARC . This is *part three* of the algorithm .

To remember which paths we have already added , we consider |C|+1 Boolean matrices,one for $f \in N$ denoted UNI and the others for $g \in C$ denoted MEM(g) the sizes of which will be defined later .

For every new vertex s constructed in part two it is convenient to remember its generating vertices OR(s) and EXT(s) , the suffix RD(s) which labels the path (s,EXT(s)) and the prefix RG(s) that labels the path (OR(s) ,s) .

In order to prove ,that we have to consider a finite number of paths (q,f,q') , we use the chosen properties of the system T and establish the following:

Lemma 4.1 : *In every state of the graph constructed by the algorithm ,the outdegree of a new vertex is exactly* 1 *,the indegree of a new vertex is* 1 *except for the successor of an initial vertex but then , every inedge has the same label .*

The proof goes by induction on the creation time of the vertices and uses the property: T is basic .

Corollary 4.1 : *When a path contains a non initial vertex* s *created by a path* (q,f,q') *the vertices of which are* , $s_1, s_2, \ldots s_{|g|-1}$ *,it contains the subpath of* (q,f,q') : (s_1, ,s_i) *if* s_i *is its end , otherwise* (s_1, ,q') .

This corollary and the property : T is semi-reduced , prove:

Proposition 4.1 : *Whenever* (q,f,q') *is a path of a graph in the algorithm and* f∈FUN , q *and* q' *are initial vertices .*

Corollary 4.2 : *The algorithm adds at most* $|C|.n^2.\max_{g \in C}(|g|-1)$ *vertices to the initial graph .*

Therefore we can consider one set W of $m = |C|.n^2.\max_{g \in C}(|g|-1)$ vertices, the initial vertices are a subset S ,| S|=n and m is in $O(n^2)$. The algorithm only adds edges . The queue ARC will contain at most $|A|.m^2$ edges . New terms are put in ARC when new paths (q,f,q') are found and new coefficients 1 in matrices UNI and MEM(g) the size of which are n , hence the algorithm has a finite number of steps.

5. Description of the algorithm.

Data :

The system T is defined by the sets H , N, C and

-for every f∈H ,a set PROJ(f) = {g such that (f,g)∈T }

-for every x∈A ,a set FACT(x) = {f∈HUN such that x is a letter of f}

-for every f∈ FACT(x) ,a set of couples of words

DEC(f,x) = { (FG,FD) such that FGxFD = f }

-for every g∈C an integer l(g) = |g|-1 .

The graphs are defined by a set of vertices {1,2,...m} where S={1,2..n} and |A| m.m Boolean matrices :

MAT(x)(i,j) = 1 iff (i,x,j) is an edge .

Initially we consider all the edges of G_0 .The set F of final vertices is initially the set of final states of $\mathcal{A}$.

We have also the auxillary data :

-a n.n Boolean matrix UNI ,initially UNI(i,j) = 1 iff i=j .

-for every g∈C a n.n Boolean matrix MEM(g) initially O.

-for every s>n two elements of S : OR(s) and EXT(s) ,and two words RG(s) and RD(s) ,initially they are Ø or 1 .

-a counter COMPT : an integer ≥n used for the allocation of the new edges on "new" vertices .Initially COMPT= n.

-a counter ETAP of stages of the algorithm related to a state of the graph and the order of the appearence of 1 in the matrices UNI and MEM(g) .

-a queue ARC initially contains all the edges of G_0 .

Part one of the algorithm :The search of paths (q,f,q') that contain a given edge (i,a,j) .

We choose ,for this graph algorithm ,to use a m.k matrix , the lines of which construct the sucessor of the last line ,and we short our search whenever a vertices s>n by putting directly its path-successor by corrollary 4.1 .The result is a set QxQ' of all the couples of extremities of such a path .We search Q then Q' and have to compute their cartesian product.Roughly the search of Q' can be describe by :

```
CHERCHE( { MAT(x),OR(x),EXT(x) } ,f, a, (FG,FD),j )
   k= |FD|
   S is a m.k Boolean matrix
   [for p=1 to m and p≠j do  S(0,p)=0;S(0,j);]
   [for s=1 to  p  do
      x:=front (FD); delete (FD);
      [for r=1 to  n  do
```

```
if S(s-1,r)=1  then [for t=1 to m do if  MAT(x)(r,t) =1;S(s,t)=1 ;]
[for  r=n+1  to  m  do
   if S(s-1,r) =1 then
         compare (FD and RD(r) );
         if RD(r) prefix of FD then S(s+|RD(r)|-1,EXT(r))=1;]]
```

We denote part one by : CHERCHE((i,a,j),f,(FG,FD),QxQ') .

Part two of the algorithm : The adjunction of a path (q,g,q') whenever a path (q,f,q') is found by part one with f∈H; and the adjunction of one edge in ARC .

We choose to put the extremities of new edges with respect to the order of the numbers that represent the vertices n+1, n+2 ,.. ,m by the mean of the counter COMPT . We denote this part by :

ADJ((q,q'),g,{MAT(x),OR(x) ,EXT(x),RG(x),RD(x) ,ARC)

and describe it roughly :

```
g is represented by a queue (LIST )
x:=front(LIST); delete (LIST);
z:=COMPT;
s:=COMPT +1 ;
MAT(x)(q,s) := 1; enter (ARC,(q,x,s));
COMPT:=COMPT+1; OR(s):=q;  EXT(s):=q'; RD(s):= LIST; RG(s):=x;
repeat until COMPT=l(g)+z
      x:=front (LIST);delete (LIST) ;
      MAT(x)(s,s+1) :=1;
      s:=s+1; OR(s):=q ; EXT(s):= q'; RD(s):= LIST; RG(s) :=RG(s-1)x;
      COMPT:=COMPT+1 ;
X:=front(LIST);  delete (LIST); MAT(x)(s,q') := 1 ;
```

Part three of the algorithm :The adjunction of edges (q,x,r) , whenever a path (q,f,q') is found in part one with f∈N and (q',x,r) is already an edge of the graph ; these edges are added both in the grah and in ARC .

We denote this part by CLOT((q,q'),x, MAT(x), ARC):

```
[for s=1 to m do
   if MAT(x)(q',s) =1 and MAT(x)(q,s) = 0 then
      MAT(x)(q,s) := 1;
      enter ( ARC,(q,x,s));]
```

The whole algorithm can be now simply describe :

```
    ETAP:=1
BC: if empty (ARC) then halt ;
    (i,a,j):=front ARC; delete (ARC);
    [for all f∈FACT(a) do
       [for all (FG,FD)∈DEC(f,a) do
          CHERCHE((i,a,j),f,(FG,FD),QxQ');
          [for all (q,q')∈ QxQ' do

             if f∈ N and UNI(q,q') =0 then
                   ETAP:=ETAP+1;
                   UNI(q,q'):=1;
                   if q'∈F then F:=FU{q'} ;
                   [for all x∈A do  CLOT ((q,q'),x,MAT(x),ARC);]
             else if f∈H then
                   [for all g∈PROJ(f) do
                      if MEM(g) =0 then
                         ETAP:=ETAP+1 ;
                         MEM(g)(q,q') := 1;
                         ADJ((q,q'),g,{MAT(x),OR(x),EXT(x),RG(x),RD(x)},ARC);]]]]
    go to BC;
```

6. The running time of this algorithm .

We evaluate each part of the algorithm:

-the lenght of the queue ARC is already in $O(n^4)$ but we can give a smaller bound : initially there are n^2 edges in ARC the new ones have their invertices in S so the size of ARC is in

$O(n^2)+O(n.m) = O(n^3)$.

-generally ,part one countains n.m executions of instruction in O(1) and m-n also in O(1) ; its running time is $O(n.m)+O(m-n) = O(n^3)$. If H=Ø then n=m and it becomes $O(n^2)$; if the lenght of words in FUN is smaller than 2 , O(n) . This part is called every time a new edge is put in front of ARC , at most $O(n^3)$ times .

-part two ADJ has a running time O(1) and is called every time a new 1 appears in MEM(g) , at most $|C|.n^2 = O(n^2)$ times .

-in part three each instruction costs O(1) and is called m times , the running time is $O(n^2)$ .This part is called when a new 1 appears in

UNI ,at most n^2 times .

Then the running time of the whole algorithm is:

-in general $O(n^3)O(n^3) + O(1)O(n^2) + O(n^2)O(n^2) = O(n^6)$

-when $H=\emptyset$ $O(n^2)O(n^2) + O(n)O(n^2) = O(n^4)$

-if $\max|g| \leq 1$ (monadic) $O(n^2)O(n^2) + O(1)O(n^2) + O(n)O(n^2) = O(n^4)$

-if $\max|f| \leq 2$,$f \in FUN$ $O(n)O(n^3) + O(1)O(n^2) + On^2)O(n^2) = O(n^4)$

-if $\max|g| \leq 1$ and $\max |f| \leq 2$,$f \in HUN$ $O(n)O(n^2) + O(1)O(n^2)+O(n)O(n^2)=O(n^3)$

7.The automaton represented by the last state G_t of the algorithm graph recognizes $\Delta^*(R)$.

Let G_i denotes the state of the graph just before the value of ETAP is i+1, R_i is the language recognized by the automaton $\mathcal{A}_i$ the graph of which is G_i .Eventually $G_t = G_{t+1} = G_{t+k}$ when the consideration of the last element of the queue does not create either new value 1 in the matrices UNI and MEM(g) or new edges by ADJ or CLOT .It is easy to prove

Proposition 7.1 : $R_t \subset \Delta^*(R)$.

The proof uses , properties of $\Delta_\#$, $\Delta_\#^*$ and Δ^* seen in 1 and the following lemma :

Lemma 7.1 : $R_i \subset R_{i-1} \cup \Delta_\#(R_{i-1})$.

The proof goes by induction on k defined by : given a path (q_0, u, q) in G_i , k is the the number of its edges which are not in G_{i-1} and the proof uses Corollary 4.1 .

To prove that $\Delta^*(R) \subset R_t$ we have to establish further results on the paths of G_t.

Lemma 7.2 : *Every path in* G_t *ending in* S *is factorized in subpaths ending in* S *,the first edges of which (its caracteristic edge) are in* ARC .

Lemma 7.2 : *If* (i,f,j) *is a path in* G_t *with* i *and* j *in* S *then* :

-*if* $f \in N$ *then* UNI (i,j) =1.

-*if* $f \in H$, $\forall g \in$ PROJ(f) MEM(g)(i,j) = 1 *and a path* (i,g,j) *exists in* G_t.

The proof uses Lemma7.2 and the last caracteristic edge to be put in front of ARC .

Lemma7.3 : *If* $i,j,k \in S$,$f \in N$, $h \in A^+$ *and* (i,f,j) *and* (j,h,k) *are paths in* G_t *where all the vertices of* (j,h,k) *,except its ends ,are not in* S , *then a path* (i,h,k) *exsists in* G_t .

The proof goes by induction on the value of ETAP when the caracteristic edge of (j,h,k) is created .

Then it is easy to prove that $\Delta^1(R_t) \subset R_t$ then $\Delta^*(R_t) \subset R_t$ and $\Delta^*(R) \subset \Delta^*(R_t) \subset R_t$.

8.Applications

If we consider the congruence generated by a basic ,semi-reduced, Church-Rosser system, each congruence class is defined by its minimal element and our results imply :

- -Monoids finitly presented by such systems have the cross-section property [14] and [8].
- -The two problems :
  - -Do two regular languages have a non zero intersection with a congruence class ?
  - -Does a regular language have a non zero intersection with the set of descendants of another regular language ?

  are decidable for a basic ,semi-reduced ,Church-Rosser system .

In conclusion ,we think that ,this class of system is the largest one in which Theorems 1 and 2 are true ,since Church-Rosser , semi-reduced systems exist which are not basic and in which ,sets of descendants of regular sets are not regular [2] .

References :

[1] M.Benois: Parties rationnelles du groupe libre, C.R.Acad.Sci.Paris ser. A 269,1969,1188-1190.

[2] M.Benois et P.Butzbach :Langages algébriques déterministes et transversale rationnelle ,Rapport de recherche n° 446 , L.S.D. IMAG 1984.

[3] M.Benois et J.Sakarovitch :On the complexity of some extended word problems defined by cancellation rules ,Information Processing Letters à paraître .

[4] J.Berstel :Transductions and context free langages .Teubner 1979 .

[5] R.Book , M.Jantzen and C.Wrathall , Monadic Thue Systems , Theoret. Comput. Sci. 19 (1982) 231-251 .

[6] R.V.Book and F.Otto : Cancellatio rules and extended word problems , Information Processing Letters 20,1985,5-11.

[7] D.Dolev and A.Yao : On the security of public key protocols ,I.E.E.E. Trans.Information Theory IT 29 , 1983, 198-208.

[8] S.Eilenberg : Automata , Languages , and Machines Vol A ,Academic Press , 1974.

[9] G.Huet : Confluent reductions: Abstract properties and applications to term rewriting systems , J. Assoc. Comp. Mach. 27, 1980 797-821.

[10] J.Hopcroft and J.Ullman : Introduction to Automata Theory, Langages and Computation Addison-Wesley , 1979.

[11] P.Narendran: Church -Rosser and related Thue systems,Report n° 84CRD176 ,General Electric Corporate Research and Development , Schenectady , N.Y. ,1984 .

[12] M.Nivat (et M.Benois) : Congruences parfaites et quasi parfaites , Séminaire Dubreil ,25° année ,1971-72 , 7-01-09 .

[13] J.Sakarovitch :Description des monoïdes de type fini ,Rapport n° 80-36 LITP ,1980 .

[14] J.Sakarovitch : Syntaxe des langages de Chomsky Thèse Paris7 1979.