Information Processing Letters 23 (1986) 281–287
North-Holland

3 December 1986

ON THE COMPLEXITY OF SOME EXTENDED WORD PROBLEMS DEFINED BY CANCELLATION RULES *

Michèle BENOIS

IMAG Grenoble, B.P. 68, 38402 St. Martin d'Hères Cedex, France

Jacques SAKAROVITCH

LITP, University of Paris 6, 4 Place Jussieu, 75230 Paris Cedex 05, France

Communicated by L. Boasson
Received October 1985
Revised January 1986

We give an algorithm which computes the set of descendants of a regular set R, for Thue systems of certain type. The complexity of the algorithm is $O(m^3)$ where m is the number of states of an automaton recognising R. This allows to improve the known complexity bounds for some extended word problems defined by cancellation rules.

Keywords: Computational complexity, automata theory, Thue systems

1. Introduction

Recent works on public key encryption for secure network communication have brought back to light an old result in formal language theory and set the question of its complexity.

In [5], Dolev and Yao have developed a formal model for name-stamp protocols. As reported in [3], "they proved that the question of security of name-stamp protocols is tractable by showing that a certain extended word problem related to the set of cancellation rules is tractable". It has been pointed out in [3,4,6] that the rewriting process that underlies the application of cancellation rules is the rewriting of strings by means of a Thue system of a certain type and with the Church–Rosser property. The extended word problem of Dolev and Yao then reduces to the problem of the emptiness of the intersection of a finite set of words with the set of descendants (with respect to the Thue system) of the words of a regular set.

It is known [1,2,7] that, given a Thue system of the prescribed type, the set of descendants of a regular set is a regular set (Theorem 2.1 below). Book and Otto [3] gave an algorithm where a finite automaton, with m states, recognising the set of descendants of R, is computed in $O(m^4)$ steps from a finite automaton, with m states, recognising R.

Here, we present an algorithm which performs the same task in $O(m^3)$ steps. With this algorithm we then improve the complexity bounds of some word problems that were considered in [3] and we derive a new algorithm for the transitive closure of a directed graph.

2. Definitions and results

Let Σ be a finite set called the *alphabet*; a finite sequence of elements of Σ is called a *word* over Σ. The set of all words over Σ, together with the concatenation of sequences, is the free monoid

* This work was partially supported by A.D.I. under Contract 83/695.

over Σ and is denoted by Σ^*; the *empty word*, that is the empty sequence, is the identity of Σ^* and is denoted by λ. The *length* of a word f on Σ is denoted by $|f|$; it is its length as a sequence of elements of Σ. We refer to [8] for the definitions of finite automata and of regular sets on Σ, and we follow the notation therein.

We call *Thue system* on Σ any finite subset of $\Sigma^* \times \Sigma^*$. A Thue system T defines a *reduction relation* by the following: let f and g be words on Σ; then we denote by $f \to_T g$ the following relation: there exist h and k in Σ^* and (u, v) in T such that $f = huk$ and $g = hvk$; the reduction relation denoted by $\to_T^*$ is the reflexive and transitive closure of $\to_T$. A word g is a descendant of a word f if $f \to_T^* g$ and the set of descendants of a set L of Σ^* will be denoted here, as in [3], by $\Delta_T^*(L)$:

$$\Delta_T^*(L) = \{g \in \Sigma^* \mid f \in L, f \to_T^* g\}.$$

As in [3], a Thue system T is said to be *homogeneous of degree* k if every element is of the form (u, λ), where $|u| = k$. Thus, an homogeneous Thue system of degree k is completely determined by a subset of Σ^k. The cancellation rules of [5] form homogeneous Thue systems of degree 2.

The following result has been proved in [1,2,7].

Theorem 2.1. *Let* T *be a homogeneous Thue system of degree* 2 *on* Σ. *If* R *is a regular set on* Σ, *then* $\Delta_T^*(R)$ *is also a regular set.*

Our purpose here is to give a measure of the complexity of the construction of an automaton which recognises $\Delta_T^*(R)$.

Theorem 2.2. *Let* T *be a finite Thue system homogeneous of degree* 2 *on* Σ. *If* R *is a regular set on* Σ *specified by a nondeterministic finite-state automation with* m *states, then one can effectively construct in* $O(m^3)$ *steps a nondeterministic finite-state automation of* $\Delta_T^*(R)$.

The proof of Theorem 2.2, to be found in the next section, consists of the construction of an algorithm which is the effective version of the proof of Theorem 2.1, given in [7], and which we recall here for the sake of completeness.

From now on, let Σ be a finite alphabet and T an homogeneous Thue system of degree 2 on Σ. In the statement, T will be understood, and we shall simply write $f \to g$, $f \to^* g$, $\Delta^*(R)$. We denote by θ the subset of Σ^2 which defines T, and by S the set of words on Σ which have λ in their descendants.

The following basic lemma gives a relation between a word and its descendants.

Lemma 2.3. *If* f *and* g *are words on* Σ *such that* $f \to^* g$ *and* $g = a_1 a_2 \ldots a_n$, a_i *in* Σ, *then there exist* $s_0, s_1, \ldots, s_n$ *in* S *such that* $f = s_0 a_1 s_1 \ldots a_n s_n$.

Proof. The proof of Lemma 2.3 goes by induction on the length of the reduction $f \to^* g$: the initial step is trivial and the induction step follows from the property of S: If u, v, w are in S and ab is in θ, then uavbw is in S. □

Theorem 2.1 is the obvious consequence of the following more technical result.

Proposition 2.4. *Let* R *be a regular set on* Σ *accepted by the nondeterministic finite-state automaton* $\mathscr{A} = \langle Q, \Sigma, q_0, \delta, F\rangle$. *Then* $\Delta^*(R)$ *is accepted by* $\mathscr{A}' = \langle Q, \Sigma, q_0, \delta', F'\rangle$ *where* δ' *is defined by*

$$\forall q \in Q \ \forall a \in \Sigma:$$
$$p \in \delta'(q, a) \Leftrightarrow \exists s \in S: p \in \delta(q, sa)$$

and F′ *by*

$$\forall q \in Q: \quad q \in F' \Leftrightarrow \exists s \in S \ \exists p \in F: p \in \delta(q, s).$$

Proof. Recall that the transition function δ from $Q \times \Sigma$ into 2^Q is extended to a function also called δ from $2^Q \times \Sigma^*$ into 2^Q by

$$\forall q \in Q: \quad \delta(q, \lambda) = \{q\},$$
$$\forall P \subset Q \ \forall a \in \Sigma: \quad \delta(P, a) = \bigcup_{p \in P} \delta(p, a),$$
$$\forall P \subset Q \ \forall f \in \Sigma^*: \quad \delta(P, fa) = \delta(\delta(P, f), a).$$

We further extend the domain of δ to $2^Q \times 2^{\Sigma^*}$ into 2^Q by

$$\forall P \subset Q \ \forall L \subset \Sigma^*: \quad \delta(P, L) = \bigcup_{f \in L} \delta(P, f).$$

The definition of δ' may then be written as

$$\delta'(P, a) = \delta(P, Sa) \tag{1}$$

and the definition of F′ as

$$F' = \{p \in Q \mid \delta(p, S) \cap F \neq \emptyset\}. \quad (2)$$

By induction on n one has

$$\delta'(P, a_1a_2 \ldots a_n) = \delta(P, Sa_1Sa_2 \ldots Sa_nS). \quad (3)$$

The basis of the induction is indeed exactly (1); the induction step is proved as follows:

$$\begin{aligned}
&\delta'(P, a_1a_2 \ldots a_n)\\
&\quad = \delta'(\delta'(P, a_1a_2 \ldots a_{n-1}), a_n) \quad \text{by definition}\\
&\quad = \delta'(\delta(P, Sa_1Sa_2 \ldots Sa_{n-1}), a_n) \quad \text{by induction}\\
&\quad = \delta(\delta(P, Sa_1Sa_2 \ldots Sa_{n-1}), Sa_n) \quad \text{by (1)}\\
&\quad = \delta(P, Sa_1Sa_2 \ldots Sa_nS) \quad \text{by definition.}
\end{aligned}$$

Let $g = a_1a_2 \ldots a_n$ be a word on Σ; then (3) gives

$$\delta'(q_0, g) = \delta(q_0, Sa_1Sa_2 \ldots Sa_nS)$$

and (2) gives

$$\begin{aligned}
&\delta'(q_0, g) \cup F' \neq \emptyset\\
&\quad \Leftrightarrow \delta(\delta(q_0, Sa_1Sa_2 \ldots Sa_nS), S) \cap F \neq \emptyset,
\end{aligned}$$

i.e.,

$$\Leftrightarrow \delta(q_0, Sa_1Sa_2 \ldots Sa_nS) \cap F \neq \emptyset.$$

Then g is recognised by $\mathscr{A}'$ iff $Sa_1Sa_2 \ldots Sa_nS$ contains a word recognised by $\mathscr{A}$ which is equivalent, by Lemma 2.3, to the fact that g belongs to $\Delta^*(R)$. □

3. The algorithm

In the preceding section we have defined finite automata by means of transition functions. Here, we shall use two other representations of automata: the matrix representation, in order to write the algorithm, and the graph representation, in order to explain how the algorithm works.

If $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ is a finite automaton with m states, then δ may be defined by the Boolean m × m matrices M(a), for every a in Σ:

$$M(a)(i, j) = \textbf{true} \quad \text{iff} \quad j \in \delta(i, a)$$

and F may be defined by the m × 1 matrix U:

$$U(i) = \textbf{true} \quad \text{iff} \quad i \in F.$$

Then, Proposition 2.4 may be restated as follows.

Proposition 3.1. *Let* SM *be the Boolean* m × m *matrix defined by*

$$SM(i, j) = \textbf{true} \quad \textit{iff} \quad \exists s \in S: j \in \delta(i, s).$$

Then the automaton $\mathscr{A}'$ *defined by the matrices*

$$M'(a) = SM \times M(a), \qquad U' = SM \times U$$

accepts $\Delta^*(R)$.

The automaton $\mathscr{A}$ can also be represented by a labelled graph: a triple (p, a, q) is an *edge* of $\mathscr{A}$ if p, q are in Q, a is in Σ and $q \in \delta(p, a)$; it follows that (p, f, q) is a *path* in $\mathscr{A}$ if f is in Σ^* and $q \in \delta(p, f)$.

The true entries of the matrix SM represent the paths of $\mathscr{A}$ which are labelled by words in S. The key feature of the algorithm is the intertwinning of the computation of the entries of SM and of the entries of M′(a).

The data of the algorithm are:

- A set ALPHABET of the symbols of Σ.
- A set STATE of the states of $\mathscr{A}$ and $\mathscr{A}'$, of cardinal N.
- A set FIN of integers which initially represents F.
- For every a in ALPHABET, a Boolean matrix MAT(a) representing M(a).
- For every a in ALPHABET, a set RIGHT(a) (respectively LEFT(a)) of b in ALPHABET such that ab is in θ (respectively ba in θ).

A preprocessing:

(a) initialises a Boolean N × N matrix Z whose entries are initially set to **true** iff i = j,

(b) computes a queue EDGE of the edges (p, x, q) of $\mathscr{A}$.

Remark 3.2. The time necessary for the preprocessing is $O(N^2)$ for Z, and $O(N^2)$ for EDGE, hence $O(N^2)$.

The outputs of the algorithm are the Boolean matrices MAT(a) for every a in Σ.

Algorithm

(1) stage := 0

(2) LOOP: **if empty**(EDGE) **then halt**;

```
(3) (i, a, j) := front(EDGE); delete(EDGE);
      ( * (i, a, j) is the topmost triple on
       EDGE, which is now deleted from EDGE * )
(4) for all q ∈ STATE do
(5)   begin
(6)   for all b ∈ RIGHT(a) do
(7)   if MAT(b)(j, q) = true and Z(i, q) = false do
(8)     begin
(9)     stage := stage + 1;
(10)    Z(i, q) := true;
(11)    if q ∈ FIN then FIN := FIN ∪ {i};
(12)    for all x ∈ ALPHABET do
(13)    CLOSE(i, q, STATE, x, MAT(x), EDGE)
(14)    end;
(15)  for all b ∈ LEFT(a) do
(16)  if MAT(b)(q, i) = true and Z(q, j) = false do
(17)    begin
(18)    stage := stage + 1;
(19)    Z(q, j) := true;
(20)    if j ∈ FIN then FIN := FIN ∪ {q};
(21)    for all x ∈ ALPHABET do
(22)    CLOSE(q, j, STATE, x, MAT(x), EDGE)
(23)    end
(24)  end;
(25) go to LOOP;

     CLOSE(i, j, ETAT, x, M, LIST):
(1)  begin
(2)  for all p ∈ ETAT do
(3)  if M(j, p) = true and M(i, p) = false then
(4)    begin
(5)    M(i, P) := true;
(6)    enter(LIST, (i, x, p))
(7)    end
(8)  end;
```

Remark 3.3. The procedure CLOSE may modify the parameters M and LIST.

Proposition 3.4. *The time of processing* EDGE *is* $O(N^3)$.

Proof. Let us evaluate the algorithm line by line:

Line (4) costs O(N) time, lines (6) and (15) cost $O(|\text{ALPHABET}|) = O(1)$ time, and the other lines obviously cost O(1) time.

Procedure CLOSE costs O(N) time, and is executed when Z(i, j) becomes **true**, hence at most N^2 times. Elsewhere, the entries in LOOP cost O(N) time and are realised at most N^2 times (the length of the queue is $O(N^2)$). The running time is

$$O(N^2) + O(N) \times N^2 + O(N) \times O(N^2) = O(N^3).$$

□

To prove Theorem 2.2 it remains to show that the algorithm correctly specifies $\mathscr{A}'$, i.e., that MAT(a) = M'(a) for every a in Σ at the end of the algorithm (where the M'(a) are the matrices defined in Proposition 3.1).

Lemma 3.5. *Let* s *be in* S. *If* $s \neq \lambda$, *then there exist* u, v *in* S *and* ab *in* θ *such that* $s = uavb$.

Proof. The proof follows by induction on the length of s:

If $|s| = 2$, the property holds since λ is in S and there exists ab in θ, such that $s = \lambda a \lambda b$.

If $|s| > 2$, there exists cd in θ such that $s \to^* cd$. By Lemma 2.3 there exist s_0, s_1, s_2 in S such that $s = s_0 c s_1 d s_2$. If $s_2 = \lambda$, then $s = s_0 c s_1 d$ is a factorisation of the required form. If $s_2 \neq \lambda$, then $|s_2| < |s| - 2$ and the induction hypothesis gives a factorisation $s_2 = u'avb$ where u', v are in S and $u = s_0 c s_1 d u'$ is also in S. □

Lemma 3.6. *Let* i, j *be two elements in* STATE. *If there exists an* s *in* S *such that* (i, s, j) *is a path in* $\mathscr{A}$, *then* Z(i, j) *is* **true** *when the algorithm ends.*

Proof. Suppose that there exist paths (i, s, j) in $\mathscr{A}$ with s in S such that Z(i, j) = **false** at the end of the algorithm, and choose such a path (i, s, j) with s of minimal length. Lemma 3.5 shows that $s = uavb$ with $|u|, |v| < |s|$. Hence, if the path (i, s, j) is of the form (i, u, i')(i', a, p)(p, v, j')(j', b, j), then, by the hypothesis of minimality of s the entries Z(i, i') and Z(p, j') are set to **true** by the algorithm. When Z(i, i') is set to **true** (at lines (10) or (19)), procedure CLOSE is called and since the edge (i', a, p) is in the graph of $\mathscr{A}$, the edge (i, a, p) is created if it did not exist before. For the same reason, the edge (p, b, j) is created.

These two edges are put into EDGE and when the last of them is processed in the algorithm, Z(i, j) is set to **true**; a contradiction. □

Lemma 3.7. *If there exist* s *in* S, x *in* Σ, *and* i *and* j *in* STATE *such that* (i, sx, j) *is a path in* $\mathscr{A}$, *then* MAT(x)(i, j) = **true** when the algorithm ends.

The proof is an easy consequence of Lemma 3.6.

Lemma 3.8. (a) *If* Z(i, j) *is* **true** *when the algorithm ends, then there exist* s *in* S *and a path* (i, s, j) *in* $\mathscr{A}$.

(b) *If* MAT(x)(i, j) *is* **true** *when the algorithm ends, then there exists an* s *in* S *such that* (i, sx, j) *is a path in* $\mathscr{A}$.

Proof. Here, we use the variable **stage** of the algorithm; it defines an ordering on the **true** entries of the matrices Z and MAT(x).

We thus prove by induction on t the following Claim which is equivalent to the lemma:

Claim: (A) *If* Z(i, j) = **true** *at* **stage** t, *then there exist* s *in* S *and a path* (i, s, j) *in* $\mathscr{A}$.

(B) *If* MAT(x)(i, j) = **true** *at* **stage** t, *then there exist* s *in* S *and a path* (i, sx, j) *in* $\mathscr{A}$.

Proof. (A) If t = 0, then

(a) Z(i, j) = **true** iff i = j, $\lambda \in S$, and (i, λ, i) is a path of $\mathscr{A}$,

(b) MAT(x)(i, j) = **true** iff (i, x, j) is an edge in $\mathscr{A}$;

thus, the Claim is satisfied for t = 0.

If Z(i, j) is set to **true** when **stage** = t:

- either there exist:
  - an edge (i, a, p) in EDGE and hence MAT(a)(i, p) is **true** for a value t′ < t of **stage**; by induction there exists an s_1 in S such that (i, s_1a, p) is a path in $\mathscr{A}$; and
  - a letter b such that ab $\in \theta$ and MAT(b)(p, j) is **true** for a value t″ < t of **stage** and thus there exists an s_2 in S such that (p, s_2b, j) is a path in $\mathscr{A}$;

  and then the word s_1as_2b is in S and (i, s_1as_2b, j) is a path in $\mathscr{A}$;
- or there exist:
  - an edge (p, a, j) in EDGE, and
  - a letter b such that ba $\in \theta$ and MAT(b)(i, p) is **true** and the similar conclusion follows;

  and then there exists a word s_1bs_2a in S such that (i, s_1bs_2a, j) is a path in $\mathscr{A}$.

This proves part (A) of the Claim.

(B) If MAT(x)(i, j) is set to **true** when **stage** = t, then there exists a p in STATE such that MAT(x)(p, j) is **true** at **stage** = t′ < t and Z(i, p) is **true** when **stage** = t″ $\leqslant$ t. By induction there exists an s_1 in S such that (p, s_1x, j) is a path in $\mathscr{A}$ and by the proof of part (A) of the Claim (or by the induction) there exists an s_2s_1 in S such that (i, s_2s_1x, j) is a path in $\mathscr{A}$, which proves part (B) of the Claim. □

Proof of Theorem 2.2. By Lemmas 3.6 and 3.8(a), at the end of the algorithm the matrix Z is equal to the matrix SM defined in Proposition 3.1. By Lemmas 3.7 and 3.8(b), the matrices MAT(x) are equal to the matrices SM × M(x) at the end of the algorithm and it is easy to check that at the end of the algorithm the matrix FIN is equal to the matrix SM × U. □

4. Applications

Theorem 2.2 yields improved complexity bounds for several types of word problems that were considered in [3]. Moreover, from the algorithm presented above we shall derive an $O(m^3)$ algorithm for the transitive closure of a graph with m vertices.

Let T be a Thue system on Σ. A word f of Σ^* is said to be *irreducible* for T if $\Delta_T^*(f) = \{f\}$, that is, if f has no factor u which is a left component of a pair (u, v) in T. Therefore, the set R_T of irreducible words for T is a regular set of Σ^*. The Thue system T then defines a mapping ρ_T from Σ^* into $\mathscr{P}(\Sigma^*)$: $\rho_T(f)$ is the set of the descendants of f that are irreducible. Note that, in the general case, $\rho_T(f)$ may be empty or have more than one element; if the system T is homogeneous (of any degree), then $\rho_T(f)$ is never empty but may have more than one element. If T is *Church–Rosser* (cf. [3], for instance, for the definition of the Church–Rosser property), then $\rho_T(f)$ is a singleton for every f.

Corollary 4.1. *Let* T *be an homogeneous Thue system of degree* 2 *on* Σ *and let* R *be a regular set of* Σ^* *accepted by a finite nondeterministic automaton with* m *states. Then the set* $\rho_T(R)$ *is a regular set and an automaton or* $\rho_T(R)$ *can be constructed in* $O(m^3)$ *steps.*

Proof. Since $\rho_T(R) = \Delta_T^*(R) \cap R_T$, $\rho_T(R)$ is regular. An automaton for R_T has $|\Sigma|+1$ states and can be constructed in $O(|\Sigma|^2) = O(1)$ steps. □

If the system T is homogeneous of degree 2 and Church–Rosser, the same techniques may be used to solve a generalisation of the word problem in the quotient of Σ^* by the congruence generated by T.

Corollary 4.2. *Let* T *be a Thue system on* Σ *that is homogeneous of degree* 2 *and Church–Rosser. Let* R_1 *and* R_2 *be regular sets of* Σ^* *accepted by finite nondeterministic automata with* m_1 *and* m_2 *states, respectively. The problem to decide whether there exist* f_1 *in* R_1 *and* f_2 *in* R_2 *that are equivalent modulo the congruence generated by* T *can be solved in* $O(m)$ *steps where* $m = \max\{m_1^3, m_2^3, m_1^2 \times m_2^2\}$.

Proof. Since T is Church–Rosser, there exist f_1 in R_1 and f_2 in R_2 that are equivalent modulo T iff the intersection $C = \rho_T(R_1) \cap \rho_T(R_2)$ is not empty. An automaton for $\rho_T(R_1)$ has $O(m_1)$ states, an automaton for $\rho_T(R_2)$ has $O(m_2)$ states, and standard techniques on finite automata allow to decide whether or not C is empty in $O(m_1^2 \times m_2^2)$ steps. The result then follows from Corollary 4.1. □

As reported in [3], Dolev and Yao have defined the following extended word problem for a set of cancellation rules T: given a regular set R of Σ^* and a finite set F of Σ^*, is the intersection of F with the descendants of R nonempty? Since any set of cancellation rules is an homogeneous Thue system of degree 2, the above results apply (the fact that a set of cancellation rules is Church–Rosser does not matter in that problem). We thus have the following result which improves the bounds given in [3] and [5].

Corollary 4.3. *Let* T *be an homogeneous Thue system of degree* 2 *on* Σ. *Let* R_1 *and* R_2 *be regular sets of* Σ^* *that are accepted by automata with respectively* m_1 *and* m_2 *states. The problem to decide whether or not the intersection of* R_2 *with the set of descendants of* R_1 *is empty, can be solved in* $O(m)$ *steps where* $m = \max\{m_1^3, m_1^2 \times m_2^2\}$.

The proof is done in the same way as the one of Corollary 4.2.

Let us now turn to the problem of the transitive closure of a directed graph. It is known to be solvable in $O(m^3)$ where m is the number of vertices of the graph [10]. It is remarkable that a slight modification of our algorithm, indeed a simplification of it, immediately gives the same complexity result, although the method is basically different: Warshall's algorithm [9] uses an ordering of the vertices whereas our algorithm uses an ordering of the edges.

The data of the algorithm are:
- a set VERTICES of cardinal N,
- an N × N Boolean matrix M; initially M(i, j) = **true** iff (i, j) is an edge of the graph,
- a queue EDGE which is initialised with the edges (i, j) of the graph.

```
LOOP: if empty(EDGE) then halt;
      (i, j) = front(EDGE); delete(EDGE);
      CLOSE(i, j, VERTICES, M, EDGE);
      go to LOOP;

CLOSE(i, j, SOM, P, LIST):
  begin
  for all s ∈ SOM do
  if P(j, s) = true and P(i, s) = false do
    begin
    P(i, s) := true;
    enter(LIST, (i, s));
    end
  end;
```

Acknowledgment

Dr. F. Otto has kindly read two preliminary versions of this paper and made useful comments.

References

[1] M. Benois, Parties rationnelles du groupe libre, C.R. Acad. Sci. Paris Sér. A 269 (1969) 1188–1190.

[2] J. Berstel, Transductions and Context-Free Languages (Teubner, Stuttgart, 1979).

[3] R.V. Book and F. Otto, Cancellation rules and extended word problems, Inform. Process. Lett. 20 (1985) 5–11.

[4] D. Dolev, S. Even and R. Karp, On the security of ping-pong protocols, Inform. and Control 55 (1982) 57–68.
[5] D. Dolev and A. Yao, On the security of public key protocols, IEEE Trans. Inform. Theory IT-29 (1983) 198–208.
[6] S. Even and O. Goldreich, On the security of multi-party ping-pong protocols, Proc. 24th IEEE Symp. on Foundations of Computer Science (1983) 34–39.
[7] M. Fliess, Deux applications de la représentation matricielle d'une série rationnelle non commutative, J. Algebra 19 (1971) 344–353.
[8] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages and Computation (Addison-Wesley, Reading, MA, 1979).
[9] S. Warshall, A theorem on Boolean matrices, J. ACM 9 (1962) 11–12.
[10] B. Roy, Transitivité et connexité, C.R Acad. Sci. Paris Sér. A 249 (1959) 216–218.