INFORMATION AND CONTROL **66**, 163–176 (1985)

Complexity of Some Problems from the Theory of Automata

JACQUES STERN

Département de Mathématiques, Université de Caen, Caen, France

INTRODUCTION

1.1. Given a finite alphabet Σ, the *regular events* over Σ are those accepted by a finite-state automaton. By Kleene's theorem, a subset W of Σ^* is a regular event if and only if it can be constructed from the finite-word sets by boolean operations together with concatenation and $*$-operation.

In some sense, the regular events are quite simple because they are accepted by a machine with no storage capacity. Nevertheless, in recent years, much attention has been paid to special subclasses of the class of regular events; in this paper, we shall be concerned with star-free events, events of dot-depth one and piecewise testable events.

Star-free events are constructed like regular events from the finite-word sets but with the restriction that the $*$-operation is not allowed; *events of dot-depth one* and *piecewise testable events* are star free of a very simple form and will be defined below; star-free events have been characterized in the work of Schützenberger (1965) in terms of their syntactic monoid; algebraic characterizations of the other two classes have been given (Simon, 1975; Knast, 1983).

From an algorithmic point of view, these algebraic characterizations do not yield efficient procedures because computing the syntactic monoid of a regular event given, e.g., by an automaton is obviously time consuming. In this paper, we investigate the complexity of three problems which we now describe in the style of (Garey and Johnson, 1979). We refer the reader to (Aho, Hopcroft, and Ullman, 1983; Garey and Johnson, 1979), for standard concepts of complexity theory.

PROBLEM I: Finite Automaton Aperiodicity.

Instance. A deterministic finite state automaton M with input alphabet Σ.

Question. Does M recognize a star-free event?

Comment. A monoid S is *aperiodic* if for some integer n, the equation

$$x^{n+1} = x^n$$

holds for all members x of S; Schützenberger's theorem asserts that a regular event is star-free if and only if its syntactic monoid is aperiodic.

PROBLEM II: Dot Depth One Event Recognition.

Instance. A deterministic finite state automaton M with input alphabet Σ.

Question. Does M recognize an event of dot-depth one?

Comment. An event is of *dot-depth one* if it is a boolean combination of events

$$w_0 \Sigma^* w_1 \Sigma^* \cdots w_{n-1} \Sigma^* w_n$$

where $w_0, w_1, \ldots, w_n$ are words over Σ. The syntactic semigroups of events of dot-depth one have been characterized by Knast (to appear).

PROBLEM III: Piecewise Testable Event Recognition.

Instance. A deterministic finite state automaton M wiht input alphabet Σ.

Question. Does M recognize a piecewise testable event over Σ?

Comment. An event is *piecewise testable* if it is a boolean combination of events

$$\Sigma^* a_1 \Sigma^* a_2 \cdots \Sigma^* a_n \Sigma^*$$

where $a_1, a_2, \ldots, a_n$ are elements of Σ.

1.2. Our three problems deal with an automaton M with input alphabet Σ; we let Q be the set of states of M; we also let

$$m = |Q|; \qquad s = |\Sigma|; \qquad n = ms;$$

these notations will be *fixed throughout the paper*. In order to measure the complexity of an algorithmic solution to any of these problems, we shall use the integer n as a measure of the size of a given instance. We can then state:

THEOREM 1. *Dot-depth one event recognition and piecewise testable event recognition can be solved in polynomial time.*

In sharp contrast with this result we have

THEOREM 2. (i) *Finite automaton aperiodicity can be solved in polynomial space*

(ii) *Finite automaton aperiodicity is the complement of an NP-hard problem.*

Thus, unless $P = NP$, finite automaton aperiodicity cannot be solved in polynomial time. The following is left pending by Theorem 2:

OPEN PROBLEM. Is finite automaton aperiodicity P-space complete?

1. POLYNOMIAL-TIME ALGORITHMS

1.1. Given an automaton M, it is well known that one can delete some states and identify some other ones: the resulting automaton is the *minimal* automaton (see Hopcroft and Ullman, 1979) or *reduced* automaton.

Several polynomial-time algorithms are known to determine the minimal automaton. The reader is referred to (Hopcroft and Ullman, 1979), where such an algorithm is given, with running time $O(n^2)$. A more efficient algorithm, with running time $O(n \log n)$ can be found in (Hopcroft, 1971).

From this, it follows that Problems I, II, III are respectively polynomial equivalent to the analogous problems obtained by considering only reduced automata.

From now on we assume that we are considering Problems I, II, III for minimal automata. We start with the simplest problem: piecewise testable event recognition.

1.2. The minimal automata which recognize piecewise testable events have been characterized by Simon (1975); in order to state Simon's result we need some definitions. Given an automaton M, with input alphabet Σ and set of states Q, we denote by δ its transition function; now, M can be turned into a directed graph $G(M)$ with set of vertices Q, by letting the edges be the pairs (p, q) such that there is a transition from p to q; similarly, if Γ is a subset of Σ, we can define another graph $G(M, \Gamma)$ by only considering those transitions which correspond to letters in Γ.

Now, given any directed graph G, we can consider another graph $\bar{G}$, with the same vertices called the *transitive closure* of G and whose edges are the pairs (p, q), $p \neq q$, such that there is a path from p to q. If p is a vertex of G, we let

$$C(p) = \{p\} \cup \{q \colon (p, q) \in \bar{G}\}$$

and we call $C(p)$ the *component* of p. If X is a subset of the set of vertices, then, X is *strongly connected* if, given any two members p, q of X, p belongs to $C(q)$ and q to $C(p)$; a strongly connected component (SCC) is any maximal strongly connected set of vertices; a graph is *acyclic* if all its SCC have only one element; in this case, the set of vertices is partially ordered by the relation $q \in C(p)$.

We now state Simon's result:

PROPOSITION 1.2. *Let W be a regular event and let M be the minimal automaton accepting W; W is piecewise-testable if and only if the following two conditions hold*:

(i) *$G(M)$ is a directed acyclic graph*;

(ii) *for any subset Γ of Σ, each component of $G(M, \Gamma)$ has a unique maximal state.*

1.3. The algorithm that can be designed by a straightforward application of the last proposition, seems to require the determination of all graphs $\overline{G(M, \Gamma)}$ for $\Gamma \subseteq \Sigma$ and therefore cannot run in polynomial time. In order to overcome this difficulty, we define for each state q,

$$\Sigma(q) = \{a \in \Sigma : \delta(q, a) = q\};$$

now, if $G(M)$ is acyclic, then, q is a maximal state of component C of $G(M, \Gamma)$ iff

(i) $q \in C$

(ii) $\Gamma \subseteq \Sigma(q)$.

Using this remark, it is clear that if q, q' are distinct maximal states of C then, they are also distinct maximal states of some component of $G(M, \Sigma(q) \cap \Sigma(q'))$, hence, condition (ii) of Proposition 1.2 can be restricted to those subsets Γ of the form $\Sigma(q) \cap \Sigma(q')$.

We can now describe an algorithm to solve the problem: "piecewise-testable event recognition"; the main steps of this algorithm are as follows:

(1) compute $G(M)$;

(2) compute $\overline{G(M)}$;

(3) check that $G(M)$ is acyclic;

(4) for any element q of Q, compute $\Sigma(q)$;

(5) for any pair (q, q') of distinct elements in $Q \times Q$

(a) compute $G(M, \Sigma(q) \cap \Sigma(q'))$;

(b) compute $\overline{G(M, \Sigma(q) \cap \Sigma(q'))} = \overline{G(q, q')}$;

(c) for any p in Q check that (p, q) and (p, q') cannot be both edges of $\overline{G(q, q')}$.

Warshall algorithm (see Aho, Hopcroft, and Ullman, 1983) can be used to compute the transitive closure $\bar{G}$ of a graph G; its running time is $O(m^3)$, where m is the number of vertices of G.

Recall that $s = |\Sigma|$, $m = |Q|$, $n = ms$; the running time of our algorithm is dominated by step (5); this step involves m^2 uses of Warshall algorithm; we get a final bound which is $O(n^5)$.

1.4. We now turn to events of dot-depth one. In order to design an efficient algorithm, we shall make use of a characterization of the minimal automata accepting events of dot-depth one, which is due to the author (Stern, 1985). This characterization is rather intricate and will be completely described later; we first investigate a necessary condition which will be a part of the full characterization.

DEFINITION. Let k be an integer; an automaton M is *k-stable* if, whenever p, q are states of M and w is a word of length k such that p, q, $\delta(p, w)$, $\delta(q, w)$ belong to the same SCC, then, $\delta(p, w) = \delta(q, w)$.

Now it is clear that M is k-stable iff no diagram of Fig. 1 form can exist with $|w| = k$ and $p' \neq q'$; in such a diagram an arrow with label x going from q' to p means $\delta(q', x) = p$.

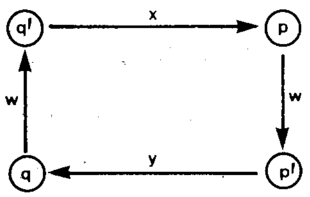
This leads us to define the following notion: two elements of $Q \times Q$ say (p, p') and (q, q') are *k-compatible* if for some word w of length k

$$\delta(p, w) = p'; \qquad \delta(q, w) = q';$$

two elements of $Q \times Q$ are *compatible* if they are k-compatible for some k. If l is an integer, then, two elements of $Q \times Q$ are $\leqslant l$-compatible if they are k-compatible for some $k \leqslant l$.

In order to determine whether or not two elements of $Q \times Q$ are k-compatible, we use a dynamic programming technique; we maintain an array

$$A[p, p', q, q']$$



$$\begin{tikzcd} q' \arrow[r, "x"] & p \arrow[d, "w"] \\ q \arrow[u, "w"] & p' \arrow[l, "y"] \end{tikzcd}$$

FIGURE 1

with boolean values indexed by Q^4. In the initial configuration $A[p, p', q, q']$ is true iff $p = p'$ and $q = q'$. The array is updated, step by step by a procedure which returns the new value B and which can be described by the following very informal pascal-like program:

```
begin
  for each $(p, p', q, q')$ in $Q^4$ do
    begin
      $B[p, p', q, q'] :=$ false;
      for each $a \in \Sigma$ do
        begin
          $p_1 := \delta(p, a)$;
          $q_1 := \delta(q, a)$;
          if $A[p_1, p', q_1, q'] =$ true then
            $B[p, p', q, q'] :=$ true
        end
    end
end
```

It is clear that after the kth step $A[p, p', q, q']$ is true iff (p, p') and (q, q') are k-compatible. Now, the updated procedure runs in time $O(sm^4)$, where $s = |\Sigma|$ and $m = |Q|$ so that the time needed to check k-compatibility is polynomially bounded in k and $n = sm$ (actually we have ignored access times needed to get, e.g., $A[p_1, p', q_1, q']$ but this does not affect the polynomial character of the computation).

An analog algorithm can be given in order to check if (p, p') and (q, q') are $\leqslant k$-compatible. As for the compatibility relation it can be handled via the following lemma:

LEMMA. *Two elements of Q^2 are compatible if and only if they are $\leqslant m^2$-compatible.*

Proof. We consider two compatible elements of Q^2 (p, p') and (q, q') and we choose a minimal integer k such that they are k-compatible; we claim that k is at most m^2. If not, we pick a word u of length k such that

$$\delta(p, u) = p'; \qquad \delta(q, u) = q'$$

and for $i = 0, \ldots, m^2$, we let u_i be the left factor of u of length i; now if

$$p_i = \delta(p, u_i); \qquad q_i = \delta(q, u_i)$$

then, one can find distinct integers $i < j$ such that

$$(p_i, q_i) = (p_j, q_j);$$

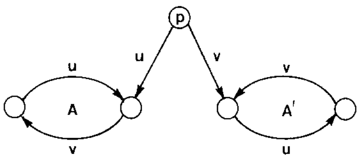


FIGURE 2

if $\bar{u}$ is obtained from u by deleting the pth letters, $i < p \leqslant j$, then it is easy to check that

$$\delta(p, \bar{u}) = p', \qquad \delta(q, \bar{u}) = q';$$

but this contradicts the minimality of k.

We close this section by outlining an algorithm which determines if an automaton M is k-stable and which runs in polynomial time (in k and n). This algorithm works as follows:

(1) compute $G(M)$;

(2) compute $\overline{G(M)}$;

(3) compute the k-compatibility relation A;

(4) for each (p, p', q, q') such that $A[p, p', q, q']$ is true and $p' \neq q'$ check that $p \notin C(q')$ or that $q \notin C(p')$

1.5. We now go on with our characterization of automata accepting events of dot-depth one. We keep the notations of the previous section: k is an integer and M is a k-stable automaton.

Two words u, v are k-coinitial (or simply coinitial if k is clear from the context) is they have the same first k letters; we write $c(u, v)$ if u and v are coinitial.

A *fork* of type (I) is a diagram of Fig. 2 form, where u, v are coinitial words and A, A' distinct SCC. A *fork* of type (II) is defined as in Fig. 3, with $c(u, x)$, $c(v, y)$, and $A \neq A'$.

Now, it is shown in (Stern, 1985) that a regular event is of dot-depth one iff for some k, its minimal automaton is k-stable and admits no fork of type (I) or (II); furthermore k can be taken to be m^3 (recall that $|Q| = m$). We already know how to check m^3-stability in polynomial time. We turn to

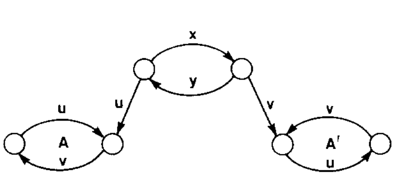


FIGURE 3

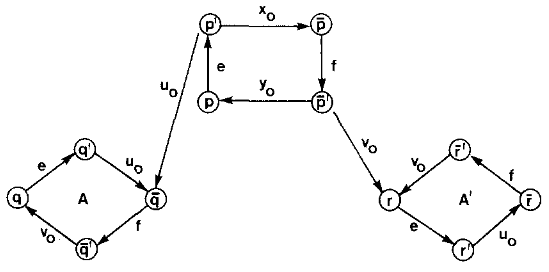


FIGURE 4

forks of type (II); the case corresponding to type (I) is similar and will be omitted. Writing $x = ex_0$, $u = eu_0$, $y = fy_0$, and $v = fv_0$ with $|e| = |f| = k$, we can get another description of forks of type (II) in Fig. 4, which in turn can be expressed by:

(i) $p, p', \bar{p}, \bar{p}'$ are in the same SCC;

(ii) q, r are *not* in the same SCC;

(iii) (p, p'), (q, q'), and (r, r') are k-compatible;

(iv) $(\bar{p}, \bar{p}')$, $(\bar{q}, \bar{q}')$, and $(\bar{r}, \bar{r}')$ are k-compatible;

(v) $(p', \bar{q})$, $(q', \bar{q})$, and $(r', \bar{r})$ are compatible;

(vi) $(\bar{p}', r)$, $(\bar{q}', q)$, and $(\bar{r}', r)$ are compatible.

As the reader has noticed, the compatibility relation cannot be the same as in Section 1.4 as it applies to triples of elements of $Q \times Q$. Nevertheless, is defined by the obvious generalization: (p, p'), (q, q'), and (r, r') are k-compatible if for some w of length k,

$$\delta(p, w) = p'; \qquad \delta(q, w) = q'; \qquad \delta(r, w) = r';$$

compatibility is defined accordingly.

Using techniques very similar to those used in Section 1.4, one can compute the k-compatibility relation and the compatibility relation in time polynomial in m and k; this yields a polynomial time algorithm to determine if a minimal automaton accepts an event of dot-depth one.

2. APERIODIC FINITE AUTOMATA

2.1. Let W be a regular event and let M be the minimal automaton accepting W; by Schützenberger's theorem, W is not star-free iff some element x of the syntactic monoid has a non-trivial period, i.e., is such that

$$\forall n \qquad x^{n+1} \neq x^n.$$

Clearly, this is equivalent to the existence of a word $u \in \Sigma^*$ and of a state p of the minimal automaton such that u defines a non trivial cycle starting at p, i.e.,

(i) $\delta(p, u) \neq p$,

(ii) for some integer r, $\delta(p, u^r) = p$.

This is pictured by Fig. 5. We note that r can be taken to be $\leqslant m$. We claim that the existence of such a diagram can be checked in polynomial space. We first provide a nondeterministic polynomial space algorithm; by Savitch's theorem (Savitch, 1970), such an algorithm can be translated into another one which is deterministic and also works in polynomial space.

The algorithm is as follows: we successively guess the letters of a word u and maintain an array A, which gives the transition function corresponding to u; at some point, we choose to stop and we check that for some state p and some integer r,

(i) $\delta(p, u) \neq p$,

(ii) $\delta(p, u^r) = p$.

This can be described in Pascal-like style:

```
begin
    stop: = false;
    for each $q$ in $Q$ do $A[q] := q$;
    while stop = false do
        begin
            guess $a \in \Sigma$;
            guess a value for stop {true or false};
            for each $q$ in $Q$ do $A[q] := \delta(A[q], a)$
        end
    guess $p \in Q$;
    guess $r$ {an integer};
    newp: = $A[p]$;
    if newp $\neq p$ then
        for $i := 1$ to $r - 1$ do
            newp = $A$[newp];
    if newp $= p$ then write ("the language is not *-free")
end.
```

It is clear that our algorithm requires polynomial space.

2.2. We keep the notations of the previous section; we will now establish that the following problem is NP-hard.

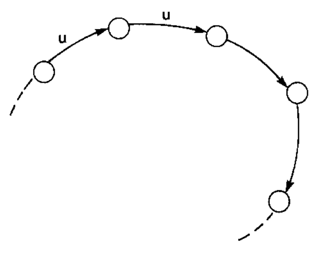


FIGURE 5

PROBLEM IV: Finite Automaton Cycle Existence.

Instance. A deterministic reduced finite automaton M with input alphabet Σ.

Question. Is there a word of Σ^* which defines a non-trivial cycle for M.

Comment. If we consider only reduced automata, this is the complement of the Problem I: Finite Automaton Aperiodicity.

We will reduce Cook's 3-SAT problem to the above; recall that an instance of SAT is a set of (disjunctive) clauses $C=(C_1,..., C_s)$ each built with exactly three literals from: $u_1, \bar{u}_1$; $u_2, \bar{u}_2$;...; $u_n, \bar{u}_n$. We assume that all clauses C_j are distinct and that $s \geqslant 2$. From C, we will define a reduced automaton M; we first describe a building block M_j of M corresponding to a given clause C_j; in order to understand the structure of the building block it is convenient to use a figure corresponding to the special case $(u_2, \bar{u}_5, u_7)$ (see Fig. 6).

The alphabet of the automaton consists of the symbols (0, 1, *); the block has an initial state I, a final state F as well as a sink state. In our final construction, the final state will be identified with the initial state of some other block; the underlying directed graph corresponding to a block is partially ordered and consists of two rows: I belongs to the upper row as well as F and, as shown by Fig. 6, it is possible to change a row only at the ith transition where u_i or $\bar{u}_i$ appears in C_j. In each row, a state can be num-

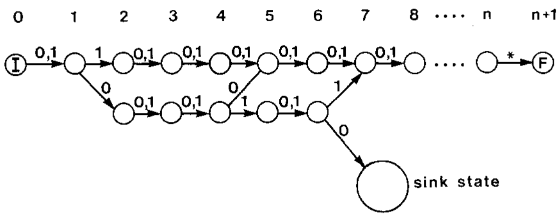


FIG. 6. Block corresponding to C_j (all transitions not mentioned go to the sink state).

bered by the number of transitions needed to reach this state starting from state I: this number will be called the position of the state. Now, any $*$ appearing before the $(n+1)$th transition carries M_j into the sink state and the block is built in such a way that a word u of length n carries M_j from I to F and only if it can be written $v*$, where v is a sequence of 0's and 1's satisfying clause C_j.

Next, we put all the blocks together by identifying the final state of M_j with the initial state of M_{j+1}. Similarly, the final state of M_s is identified with the initial state of M_1. Finally, we make all sink states the same. The resulting automaton $M(C)$ is completely specified once the accepting states are chosen; we take as accepting states all states of the upper row in each block (including the initial and final ones).

LEMMA 1. *$M(C)$ is a reduced automaton.*

Proof. We have to show that any two states p, q of $M(C)$ are not equivalent, i.e., for some word w, $\delta(p, w)$ is accepting and $\delta(q, w)$ is not (or conversely).

We make the following remarks:

(i) If p belongs to the upper row of some block and q to the lower row, then p and q cannot be equivalent (just take w to be empty).

(ii) From any state p, except the sink state, it is possible to reach the final state of the same block by some word of length $(n+1-i)$, where i is the position of p. Now, if p and q have different positions, for example, if p is in position i and q in position $i' > i$, we can take a word u of length $(n+1-i)$ which takes p to the final state of its block and it is easily seen that $\delta(q, u)$ is the sink state (because the $*$ symbol does not appear in due time).

After these remarks, we are left with the case where p, q belong to different blocks M_j and M_l, $j<l$ but are in the same position i. We pick u such that $\delta(p, u)$ is the final state of M_j. If $\delta(q, u)$ is not the final state of M_l, then, it is the sink state and we are done again. Otherwise, we note that C_{j+1} is distinct from C_{l+1} (this last clause being C_1 if $l=s$). We choose a sequence $v \in \{0, 1\}^n$ satisfying C_{j+1} and not C_{k+1} and we check that $\delta(p, uv)$ is accepting and $\delta(q, uv)$ is not. This finishes the proof of the lemma.

LEMMA 2. *If v satisfies the set of clauses C, then, $v*$ defines a nontrivial cycle in $M(C)$.*

Proof. The cycle is started at the initial state of M_1. After processing $u = v*$ j times, the final state of M_j is reached. The conclusion of Lemma 2 is now clear.

It is not possible to establish a converse of Lemma 2. The following partial converse covers a special case; subsequent arguments will show how the general case can be reduced to this special case.

LEMMA 3. *If the number s of clauses is a prime number, then, the existence of a nontrivial cycle in $M(C)$ implies the satisfiability of C.*

Proof. If $M(C)$ admits a nontrivial cycle starting at state p and defined by $u \in \Sigma^*$, then, for any left factor x of u, $\delta(p, x)$ is not the sink state. For this reason, it follows that some left factor x of u is such that $\delta(p, x_0)$ is the final state of the block of p. Writing $u = x_0 u_0$, we see that $u' = u_0 x_0$ defines a nontrivial cycle starting at the initial state I of some block M_{j_0}.

We claim that u' can be written

$$v_1 * v_2 * \cdots v_l * \qquad \text{with } v_1,..., v_l \in \{0, 1\}^n;$$

indeed, if the first letter a of u' is not 0 or 1, it carries the automaton into the sink state. Similarly, the ith letter of u' has to be 0 or 1 if $i \neq n+1 \bmod n+1$ and $*$ if $i = n+1 \bmod n+1$. Now, if u's length is not a multiple of $n+1$, the sink state is reached while processing u'. This cannot happen and the claim follows.

We observe that, for any integer t, $(u')^t$ carries I into the initial state of the block M_j, where j defined by

(i) $j \in \{1,..., s\}$

(ii) $j = j_0 + lt \bmod s$.

Now, l cannot be equal to s, modulo s otherwise $\delta(I, u')$ is I and we do not have a real cycle. As s is a prime number, l and s are relatively prime and accordingly, for any $j \in \{1,..., s\}$, there exists a t such that

$$j = j_0 + lt \bmod s;$$

if we process $(u')^{t+1}$, we reach the initial state of M_j after processing $(u')^t$. Then, we go on processing v_1 without reaching the sink state; this means that v_1 satisfies C_j. Because j can be any member of $\{1,..., s\}$, we get that v_1 satisfies C. This is exactly what was to be proved.

2.3. We will show that, given an instance C of the problem 3-SAT consisting of s clauses $C_1,..., C_s$, it is possible to add more clauses so that

(i) the resulting instance $\hat{C}$ of 3-SAT has a prime number of clauses;

(ii) C is satisfiable if and only if $\hat{C}$ is;

(iii) $\hat{C}$ can be computed from C in polynomial time.

Thus, we get from the results in the previous section:

PROPOSITION. *$M(\hat{C})$ admits a nontrivial cycle if and only if C is satisfiable.*

Therefore, the Problem IV: Finite Automaton Cycle Existence is *NP*-hard.

We now give the construction of $\hat{C}$. We assume that C has s distinct clauses, built with the literals $u_1, \bar{u}_1; u_2, \bar{u}_2;...; u_n, \bar{u}_n$, and we use the following result from number theory known as Bertrand's postulate.

THEOREM (Hardy and Wright, 1954, p. 343). *For every integer s there exists at least one prime number p such that $s < p \leqslant 2s$.*

If r is the first prime number $\geqslant s$, then we claim that r can be computed in time $O(s^{3/2})$. This is because a table of prime numbers up to $2s$, can be obtained by the standard Eratosthenes sieve method in time $O(s^{3/2})$. (The reader should not make any confusion with the usual primality problems where the running time of a test is estimated in terms of the number of digits. In our framework, we use the number itself!)

Now $\hat{C}$ can be constructed by defining

$$\hat{C}_j = C_j \qquad \text{if } j \leqslant s,$$

and

$$\hat{C}_{s+i} = (u_{n+3i+1}, u_{n+3i+2}, u_{n+3i}) \qquad \text{if } 0 < i \leqslant r - s.$$

Clearly, $\hat{C}$ has prime number of clauses and is satisfiable if and only if C is; also, $\hat{C}$ can be built in polynomial time, because this is true for the computation of r from s. This is what was to be proved.

RECEIVED: April 24, 1984; ACCEPTED: July 1, 1985

REFERENCES

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. (1983), "Data Structures and Algorithms," Addison–Wesley, Reading, Mass.

COOK, S. A. (1900), The complexity of theorem proving procedures, *in* "Proceedings, 3rd Ann. ACM Sympos. on Theory of Computing," pp. 151–158, Assoc. Comput. Mach., New York.

EILENBERG, S. (1976), "Automata, Languages and Machines," Vols. A and B, Academic Press, New York.

GAREY, M. R., AND JOHNSON, D. S. (1979), "Computers and Intractability, A Guide to the Theory of *NP*-Completeness," Freeman, San Franciso.

HARDY, G. H., AND WRIGHT, E. M. (1954), "An Introduction to the Theory of Numbers," 3rd ed., Oxford Univ. Press, London.

HOPCROFT, J. E. (1971), An $n \log n$ algorithm for minimizing states in a finite automaton, *in* "Theory and Machines and Computations," (Z. Kohavi and A. Paz, Eds.), Academic Press, New York.

HOPCROFT, J. E., AND ULLMAN, J. D. (1979), "Introduction to Automata Theory, Languages and Computations," Addison–Wesley, Reading, Mass.

KNAST (1983), A semigroup characterization of dot-depth one languages, *Rairo Inform. Théor.* **17**, 321–330.

SAVITCH, W. J. (1970), Relationship between nondeterministic and deterministic tape complexities, *J. Comput. System. Sci.* **4**, 177–192.

SCHÜTZENBERGER, M. P. (1965), On finite monoids having only trivial subgroups, *Inform. Contr.* **8**, 190–194.

SIMON, I. (1975), Piecewise testable events, Lecture notes in Computer Science Vol. 33, pp. 214–222, Springer–Verlag, Berlin.

STERN, J. (1985), Characterizations of some classes of regular events. *Theoret. Comput. Sci.* **35**, 17–42.