

# Finite-automaton aperiodicity is PSPACE-complete\*

Sang Cho and Dung T. Huynh

*Computer Science Program, University of Texas at Dallas, Richardson, Texas 75083, USA*

Communicated by D. Perrin

Received March 1989

Revised July 1989

## *Abstract*

Cho, S. and D.T. Huynh, Finite-automaton aperiodicity is PSPACE-complete, *Theoretical Computer Science* 88 (1991) 99–116.

In this paper, we solve an open problem raised by Stern (1985) – “Is finite-automaton aperiodicity PSPACE-complete?” – by providing an affirmative answer. We also characterize the exact complexity of two other problems considered by Stern: (1) dot-depth-one language recognition and (2) piecewise testable language recognition. We show that these two problems are logspace-complete for NL (the class of languages accepted by nondeterministic logspace-bounded Turing machines).

## 0. Introduction

In a paper [9] entitled “Complexity of some problems from the theory of automata,” Stern investigated the complexity of three problems: (1) finite-automaton aperiodicity, (2) dot-depth-one language recognition and (3) piecewise testable language recognition. In that paper, one can find polynomial-time algorithms for (2), (3), a polynomial-space algorithm for (1), and a proof that (1) is CoNP-hard. Since there is a gap between the upper and lower bounds of finite-automaton aperiodicity, the author raised the question “Is finite-automaton aperiodicity PSPACE-complete?”. We will show that finite-automaton aperiodicity is indeed PSPACE-complete. We will also characterize the exact complexity of (2) and (3) by showing that these two problems are logspace-complete for NL. (The reader is assumed to be familiar with basic complexity-theoretic notions that can be found in [3].)

\* This research was partially supported by the National Science Foundation under Grant DCR-8696097 and by the Organized Research Awards Program of the University of Texas at Dallas.

In the sequel we provide the necessary definitions. Given a finite alphabet  $\Sigma$ , the regular languages over  $\Sigma$  are those accepted by a finite-state automata. Regular languages can be constructed from the finite sets of strings by Boolean operations (union and complement) together with concatenation and  $*$ -operation. *Star-free languages* are constructed like regular languages from the finite sets of strings but with the restriction that the  $*$ -operation is not allowed; languages of dot-depth-one and piecewise testable languages are star-free of a simple form and are defined as follows.

A language is of *dot-depth one* if it is a Boolean combination of languages

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

where  $w_0, w_1, \dots, w_n$  are strings over  $\Sigma$ .

A language is *piecewise testable* if it is a Boolean combination of languages

$$\Sigma^* a_1 \Sigma^* a_2 \dots \Sigma^* a_n \Sigma^*,$$

where  $a_1, a_2, \dots, a_n$  are elements of  $\Sigma$ .

We now introduce the formal definitions of the three problems (1), (2) and (3) mentioned above. Finite-automaton aperiodicity is defined as follows:

*Instance.* A minimum-state deterministic finite-state automaton DFA  $M$  with input alphabet  $\Sigma$ .

*Question.* Does  $M$  recognize a star-free event?

Dot-depth-one language recognition is defined as follows:

*Instance.* A minimum-state DFA  $M$  with input alphabet  $\Sigma$ .

*Question.* Does  $M$  recognize a language of dot-depth one?

Piecewise testable language recognition is defined as follows:

*Instance.* A minimum-state DFA  $M$  with input alphabet  $\Sigma$ .

*Question.* Does  $M$  recognize a piecewise testable language?

## 1. Finite-automaton aperiodicity is PSPACE-complete

In this section we will show the main result of this paper; namely, that finite-automaton aperiodicity is PSPACE-complete. We first introduce a condition that characterizes the star-free languages.

**Proposition 1.1(a)** (Schützenberger [7]). *A regular language  $W \subseteq \Sigma^*$  is star-free iff  $W$  is aperiodic, i.e. for all element  $x$  of the syntactic monoid there is some integer  $n$  such that  $x^{n+1} = x^n$ .*

Thus, a regular language  $W$  is not star-free iff some element  $x$  of the syntactic monoid has a nontrivial period, i.e. for all  $n$ ,  $x^{n+1} \neq x^n$ . This condition can be stated in terms of minimum-state DFAs as follows.

**Proposition 1.1(b).** *A regular language accepted by a minimum-state DFA  $M$  is not star-free iff there is a word  $u \in \Sigma^*$  and a state  $p$  such that  $u$  defines a nontrivial cycle starting at  $p$ , i.e. (1)  $\delta(p, u) \neq p$  and (2) for some positive integer  $r$ ,  $\delta(p, u^r) = p$ .*

The following problem is the complement of finite-automaton aperiodicity.

**Definition 1.2.** Finite-automaton cycle existence is defined as follows:

*Instance.* A minimum-state DFA  $M$  with input alphabet  $\Sigma$ .

*Question.* Is there a word  $u$  of  $\Sigma^*$  that defines a nontrivial cycle of  $M$ ?

Next we introduce a PSPACE-complete problem which we use to prove the PSPACE-hardness of finite-automaton cycle existence.

**Definition 1.3.** Finite-state automata intersection is the following problem:

*Instance.* A sequence  $A_1, A_2, \dots, A_n$  of DFAs having the same input alphabet  $\Sigma$ .

*Question.* Is there a string  $x \in \Sigma^*$  accepted by each of  $A_i$ ,  $1 \leq i \leq n$ ?

It was shown in [5] that finite-state automata intersection is PSPACE-complete. Since the details of the construction will be needed later in our proof of the PSPACE-hardness of finite-automaton cycle existence, we reproduce them here.

**Lemma 1.4.** [5] *Finite-state automata intersection is PSPACE-complete.*

**Proof.** It is easy to see that finite-state automata intersection is in nondeterministic linear space. Thus, by Savitch's result [6], the problem is in PSPACE. Next we reduce an arbitrary problem in PSPACE to finite-state automata intersection. To this end, let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  be a single-tape deterministic  $p(n)$  space bounded Turing machine, where  $p$  is some fixed polynomial and  $B \in \Gamma$  denotes the blank symbol. Let  $x \in \Sigma^*$  be an input string and let  $n = |x|$ . Let  $\Delta = (Q \cup \{\varepsilon\}) \times (\Sigma \cup \Gamma)$ . A string  $\text{VALCOMP}_x = \# \text{ID}_0 \# \text{ID}_1 \# \dots \# \text{ID}_m \# \# \in (\Delta \cup \{ \# \})^*$  represents a valid computation of  $M$  on input  $x$  if the following conditions are satisfied:

- (1) each  $\text{ID}_i$  is an instantaneous description of  $M$  consisting of  $M$ 's tape content (padded out to length  $p(n)$  with  $B$ 's), the position of  $M$ 's head, and the state of  $M$ ;
- (2) each  $\text{ID}_{i+1}$  follows from  $\text{ID}_i$  in one step according to the transition rules of  $M$ ;
- (3)  $\text{ID}_0$  is the start configuration of  $M$  on input  $x$  and  $\text{ID}_m$  is an accepting configuration.

Clearly,  $M$  accepts  $x \in \Sigma^*$  if and only if there is a valid computation  $\text{VALCOMP}_x = \# \text{ID}_0 \# \text{ID}_1 \# \dots \# \text{ID}_m \# \# \in (\Delta \cup \{ \# \})^*$  of  $M$  on input  $x$ . We can construct a collection of DFAs with input alphabet  $\Delta \cup \{ \# \}$  so that the intersection of the languages accepted by these DFAs will be the singleton set consisting of the string  $\text{VALCOMP}_x$  if it exists, and  $\emptyset$  otherwise.

Without loss of generality, assume that  $M$  always takes an even number of steps, and has a unique accepting state  $q_{acc}$ . Further,  $M$  erases its tape before accepting and has the head at the left end of the tape in an accepting configuration. We construct a DFA  $A_{ID}$  which checks that each  $ID_i$  is indeed an instantaneous description, i.e.  $A_{ID}$  accepts the set of strings in  $(\# \Delta^{p(n)} \# \Delta^{p(n)} \#)^*$ , so that each string  $\Delta^{p(n)}$  is of the form  $[\varepsilon, X_1][\varepsilon, X_2] \dots [\varepsilon, X_{i-1}][q, X_i][\varepsilon, X_{i+1}] \dots [\varepsilon, X_{p(n)}]$ , where  $X_i \in \Sigma \cup \Gamma$ ,  $1 \leq i \leq p(n)$  and  $q \in Q$ . In other words,  $A_{ID}$  checks that there are an even number of ID's each of length  $p(n)$  and that there is exactly one cell which contains the position of the head and the current state of  $M$  among the  $p(n)$  cells for each ID.

Next we construct two groups of DFAs to check that each  $ID_{j+1}$  follows from  $ID_j$  in one step according to the transition rules of  $M$ . Recall that given the  $(i-1)$ st,  $i$ th and  $(i+1)$ st symbols of  $ID_j$  the  $i$ th symbol of  $ID_{j+1}$  can be determined from the transition rules of  $M$ . We construct a DFA  $A_i^{even}$  which accepts strings in sets of the form  $(\# \Delta^{i-2} a_1 a_2 a_3 \Delta^{p(n)-i-1} \# \Delta^{i-2} b_1 b_2 b_3 \Delta^{p(n)-i-1})^* \# \#$  so that  $b_2$  follow from  $a_1 a_2 a_3$  according to the transition rule of  $M$ , where  $a_k, b_k \in \Delta$ ,  $1 \leq k \leq 3$ .  $A_i^{even}$  checks whether the  $i$ th symbol of  $ID_{j+1}$  follows from the  $(i-1)$ st,  $i$ th and  $(i+1)$ st symbols of  $ID_j$  for even  $j$ 's. For  $i=2$ ,  $A_2^{even}$  checks that the 1st and 2nd symbols of  $ID_{j+1}$  follow from the 1st, 2nd and 3rd symbols of  $ID_j$  for even  $j$ 's. For  $i=p(n)-1$ ,  $A_{p(n)-1}^{even}$  checks that the  $(p(n)-1)$ st and  $p(n)$ th symbols of  $ID_{j+1}$  follow from the  $(p(n)-2)$ nd,  $(p(n)-1)$ st and  $p(n)$ th symbols of  $ID_j$  for even  $j$ 's. The structure of  $A_i^{even}$  is illustrated in Fig. 1, where the states of  $A_i^{even}$  are numbered in such a way that the number assigned to a state indicates its "distance" from state  $s_i$ . Further,  $d_i$  denotes the dead state and  $f_i$  the final state of  $A_i^{even}$ . (Note that only states with the same distance can be equivalent.) From the simple structure of  $A_i^{even}$ , one can easily see that the minimum-state DFA  $A_i^{even}$  can be constructed by a deterministic logspace-bounded Turing machine.

Similarly, we construct a DFA  $A_i^{odd}$  which accepts strings in sets of the form  $\# \Delta^{p(n)} (\# \Delta^{i-2} a_1 a_2 a_3 \Delta^{p(n)-i-1} \# \Delta^{i-2} b_1 b_2 b_3 \Delta^{p(n)-i-1})^* \# \Delta^{p(n)} \# \#$ , so that  $b_2$

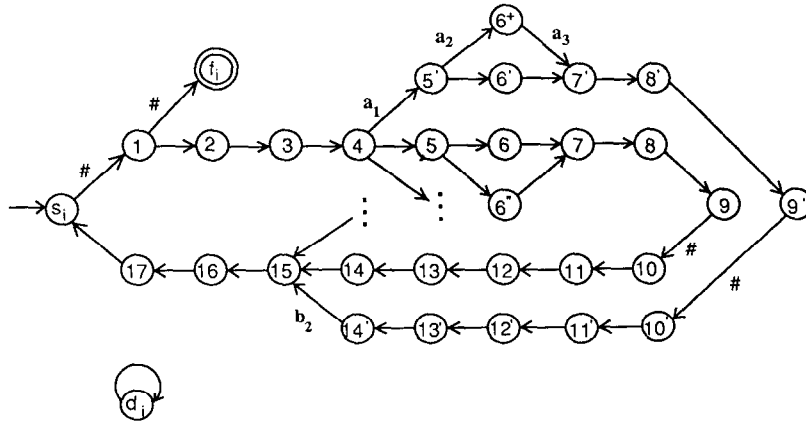


Fig. 1

Finally, we construct a DFA  $A_{\text{ends}}$  which checks that  $\text{ID}_0$  is the start configuration of the machine  $M$  and the last instantaneous description  $\text{ID}_m$  is an accepting configuration of  $M$  which is of the form  $[q_{\text{acc}}, B][\varepsilon, B] \dots [\varepsilon, B]$ . It is not hard to see that  $L(A_{\text{ID}}) \cap L(A_{\text{ends}}) \cap \bigcap_{i=2}^{p(n)-1} (L(A_i^{\text{even}}) \cap L(A_i^{\text{odd}}))$  is nonempty iff  $M$  accepts  $x$ . Note that the above reduction can be easily carried out by a deterministic logspace-bounded Turing machine. This completes the proof of Lemma 1.4.  $\square$

**Lemma 1.5** (Stern [9]). *Finite-automaton cycle existence in PSPACE.*

Fig. 2

finite-automaton cycle existence. To this end, we reconsider the DFAs constructed in the proof of Lemma 1.4. Let

$$A_1 = A_{\text{ends}} = (Q_1, \Sigma, \delta_1, s_1, \{f_1\}),$$

$$A_2 = A_{\text{ID}} = (Q_2, \Sigma, \delta_2, s_2, \{f_2\}),$$

and for  $2 \leq i \leq p(n) - 1$

$$A_{2i-1} = A_i^{\text{odd}} = (Q_{2i-1}, \Sigma, \delta_{2i-1}, s_{2i-1}, \{f_{2i-1}\}),$$

$$A_{2i} = A_i^{\text{even}} = (Q_{2i}, \Sigma, \delta_{2i}, s_{2i}, \{f_{2i}\}).$$

Without loss of generality, assume that  $Q_i \cap Q_j = \emptyset$  if  $i \neq j$  and all  $A_i$ 's are minimum-state DFAs. We construct a DFA  $A = (Q, \Sigma, \delta, s, \{f\})$  as follows:

$$Q = \{d\} \cup \bigcup_{i=1}^{2p(n)-2} (Q_i - \{d_i\}),$$

where  $d_i$  is the unique dead state of  $A_i$ ,

$$s = s_1, f = f_1, \Sigma = \Delta \cup \{\#\},$$

and  $\delta$  is defined by

- (1)  $\delta(q, a) = \delta_i(q, a)$  for  $q \in Q_i - \{r_i\}$ ,  $a \in \Sigma$  except when defined by (2),
- (2)  $\delta(f_i, \#) = s_{i+1}$  for  $1 \leq i \leq 2p(n) - 3$ ,  $\delta(f_{2p(n)-2}, \#) = s_1$ ,
- (3)  $\delta(q, a) = d$  for all  $(q, a)$  not defined by (1) and (2),

where  $d$  is the dead state of  $A$ . The structure of  $A$  is depicted in Fig. 3.

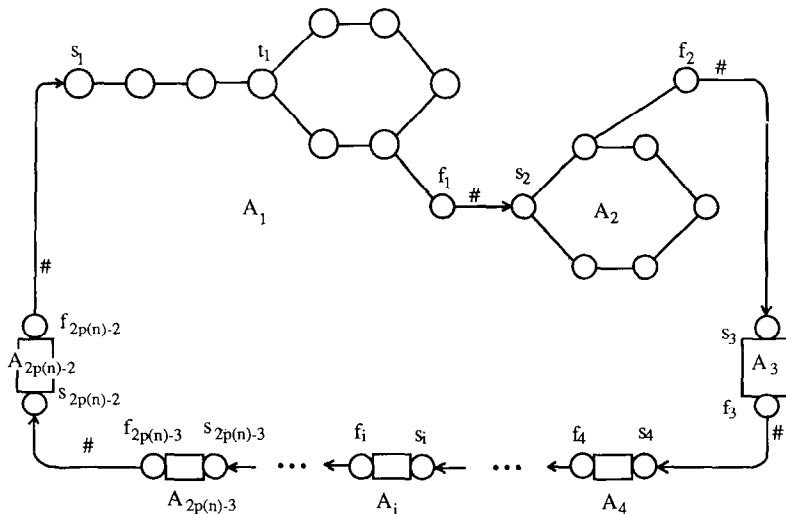


Fig. 3

Recall that in the DFA  $A$ , two states  $p, q$  are inequivalent iff there is a string  $w$  so that exactly one of  $\delta(p, w), \delta(q, w)$  is the final state  $f$ . We can easily verify that every pair of states in  $A$  are inequivalent. Thus, the DFA  $A$  is a minimum-state DFA.

Now observe that if there is a string  $x \in \Sigma^*$  accepted by  $A_1, A_2, \dots, A_{2p(n)-2}$  simultaneously, then the string  $x\#$  defines a nontrivial cycle for  $A$ . However, the converse is not necessarily true. In fact, if there is a string  $w$  that defines a nontrivial cycle for  $A$ , we cannot conclude that there is some string accepted by all  $A_i$ 's. Indeed any one of  $A_i$  may have nontrivial cycle by itself. The problem is how to eliminate nontrivial cycle from each component  $A_i$ . The solution is quite simple. To illustrate the idea, let us consider the following example.

**Example 1.6.** Consider a DFA  $M = (Q, \Sigma, \delta, q_1, \{q_1\})$ , where  $Q = \{q_1, q_2, q_3, q_4, q_5\}$ ,  $\Sigma = \{a, b\}$  and  $\delta$  is defined as

$$\delta(q_1, a) = q_2, \quad \delta(q_2, b) = q_3, \quad \delta(q_3, a) = q_4, \quad \delta(q_4, b) = q_1,$$

$$\delta(q, c) = q_5 \quad \text{for all } (q, c) \text{ not defined above, where } q_5 \text{ is the dead state.}$$

Clearly,  $M$  is a minimum-state DFA and the string  $ab$  defines a nontrivial cycle for  $M$ . However, we can modify  $M$  by expanding its input alphabet so that there is no nontrivial cycle for the modified DFA.

Let  $M' = (Q, \Sigma', \delta', q_1, \{q_1\})$  where  $\Sigma' = \Sigma \times \{0, 1, 2, 3\}$  and  $\delta'$  is defined by  $\delta'(q, \langle c, i \rangle) = p$ , where  $\delta(q, c) = p$  and  $i$  is the distance of  $q$  from  $q_1$ , i.e.  $i$  is the length of some shortest string  $x$  such that  $\delta(q_1, x) = q$ ; otherwise,  $\delta'(q, \langle c, i \rangle) = q_5$ . Then, clearly,  $M'$  is a minimum-state DFA and there is no nontrivial cycle for  $M'$ . (The construction of  $M'$  is illustrated in Fig. 4.)

We apply the above idea to eliminate nontrivial cycles from each DFA  $A_i$ . Note that all the cycles in  $A_i$  are of length  $2p(n)+2$  except loops at the dead states  $d_i$ 's. Therefore, we expand the alphabet  $\Sigma$  to  $\Sigma \times \{0, \dots, 2p(n)+1\}$ . Before modifying  $A_i$ 's we need the following definition.

The *distance* of a state  $q$  in the DFA  $A_i$  is defined to be  $|x| \bmod 2p(n)+2$ , where  $x$  is a shortest string such that  $\delta_i(s_i, x) = q$ .

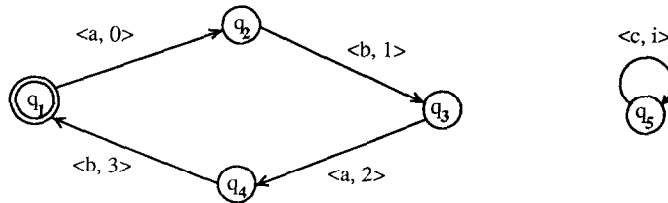


Fig. 4

For the minimum-state DFA  $A_i = (Q_i, \Sigma, \delta_i, s_i, \{f_i\})$ , we construct a minimum-state DFA  $B_i = (Q_i, \Sigma', \delta'_i, s_i, \{f_i\})$  as follows:

$$\Sigma' = \Sigma \times \{0, \dots, 2p(n) + 1\}$$

and  $\delta'_i$  is defined by

$$\delta'_i(q, \langle a, j \rangle) = \begin{cases} p & \text{if } \delta_i(q, a) = p \text{ and } j \text{ is the distance of } q, \\ d_i & \text{otherwise, where } d_i \text{ is the dead state of } B_i. \end{cases}$$

Before proving that there is no nontrivial cycle within  $B_i$ , for technical convenience, we want to classify  $B_i$ 's into two classes. The first class contains exactly all  $B_i$ 's with even  $i$  and is called the even class. The second class contains all  $B_i$ 's with odd  $i$  and is called the odd class. We can easily verify the following facts.

**Fact 1.7.** *If  $B_i$  belongs to the even class, then  $s_i$  is the only state with distance 0.*

**Fact 1.8.** *If  $B_i$  belongs to the odd class, then there is only one state  $t_i$  with distance  $p(n) + 1$ .*

We now show that there is no nontrivial cycle within  $B_i$ .

**Lemma 1.9.** *There is no nontrivial cycle within  $B_i$ .*

**Proof.** Suppose there is a nontrivial cycle within  $B_i$ . Let  $\delta'_i(p, x) = q$ ,  $p \neq q$  and  $\delta'_i(p, x') = p$ . By construction of  $B_i$  from  $A_i$  using the extension of  $\Sigma$ , the distance of  $p$  and the distance of  $q$  are identical. Thus,  $|x| \bmod 2p(n) + 2 = 0$ . If  $B_i$  belongs to the even class, then there are strings  $x_1, x_2$  so that  $x = x_1 x_2$  and  $\delta'_i(p, x_1) = s_i$ ,  $\delta'_i(s_i, x_2) = q$  since  $|x| > 0$ . Clearly,  $\delta'_i(q, x_1) = s_i$  and  $\delta'_i(s_i, x_2 x_1) = s_i$ . Then,  $\delta'_i(p, x') = \delta'_i(p, x_1 (x_2 x_1)^{r-1} x_2) = \delta'_i(s_i, (x_2 x_1)^{r-1} x_2) = \delta'_i(s_i, x_2) = q$ , which is a contradiction. Similarly, if  $B_i$  belongs to the odd class, then there are strings  $x_1, x_2$  so that  $x = x_1 x_2$  and  $\delta'_i(p, x_1) = t_i$ ,  $\delta'_i(t_i, x_2) = q$ . Clearly,  $\delta'_i(q, x_1) = t_i$  and  $\delta'_i(t_i, x_2 x_1) = t_i$ . Then,  $\delta'_i(p, x') = \delta'_i(p, x_1 (x_2 x_1)^{r-1} x_2) = \delta'_i(t_i, (x_2 x_1)^{r-1} x_2) = \delta'_i(t_i, x_2) = q$ , which is again a contradiction. Thus, there is no nontrivial cycle for  $B_i$ .  $\square$

The outline of the proof of Lemma 1.11 follows the argument in [9]. Let  $B_i = (Q_i, \Sigma', \delta'_i, s_i, \{f_i\})$  and  $d_i$  be the unique dead state of  $B_i$ ,  $1 \leq i \leq 2p(n) - 2$ . Let PRIME be the smallest prime number which is greater than  $2p(n) - 2$ . The following proposition is well known.

**Proposition 1.10** (Hardy [2]). *For any positive integer  $n$  there is at least one prime number  $p$  such that  $n < p \leq 2n$ . Furthermore,  $p$  can be computed in  $\log n$  space.*



Now for each  $1 \leq i \leq 2p(n) - 2$ , let  $B_{2p(n)-2+i}$  be a new copy of  $B_i$  such that the sets of states are all pairwise disjoint. We construct a new DFA  $B = (Q, \Sigma', \delta', s, \{f\})$  as follows:

$$Q = \{d\} \cup \bigcup_{i=1}^{\text{PRIME}} (Q_i - \{d_i\}), \quad s = s_1, \quad f = f_1,$$

$$\Sigma' = \Sigma \times \{0, \dots, 2p(n) + 1\}$$

and  $\delta'$  is defined as follows:

$$(1) \quad \delta'(q, \langle a, j \rangle) = \delta'_i(q, \langle a, j \rangle) \quad \text{for all } a \in \Sigma,$$

where  $q \in Q_i - \{d_i\}$  except when defined by (2);

$$(2) \quad \delta'(f_i, \langle \#, 2 \rangle) = s_{i+1}, \quad 1 \leq i \leq \text{PRIME} - 1,$$

$$\delta'(f_{\text{PRIME}}, \langle \#, 2 \rangle) = s_1;$$

$$(3) \quad \delta'(q, \langle a, j \rangle) = d \quad \text{if not defined by (1) and (2),}$$

where  $d$  is the dead state of  $B$ .

Clearly, the DFA  $B$  is a minimum-state DFA. We now prove the following lemma.

**Lemma 1.11.** *If  $B$  has a nontrivial cycle, then there is a string  $x$  accepted by all  $B_i$ 's and, hence, a string  $y$  accepted by all  $A_i$ 's.*

**Proof.** Suppose there is a nontrivial cycle for  $B$ . By Lemma 1.9, this cannot be a cycle within any  $B_i$ . Let  $\delta'(p, u) = q$ ,  $p \neq q$  and  $\delta'(p, u^r) = p$ . Further, let  $r$  be the smallest number satisfying the condition. Let

$$p \in Q_i - \{d_i\} \quad \text{and} \quad q \in Q_j - \{d_j\}.$$

**Fact 1.12.**  *$i \neq j$  and the distance of  $p$  in  $B_i$  and the distance of  $q$  in  $B_j$  are the same.*

**Proof of Fact 1.12.** Clearly, the distance of  $p$  in  $B_i$  and that of  $q$  in  $B_j$  are equal. Now assume, by way of contradiction, that  $i = j$ . If the computation path of  $u$  from  $p$  to  $q$  does not leave  $B_i$ , then the computation path of  $u^{r-1}$  from  $q$  to  $p$  cannot leave  $B_i$  either. This cannot happen by Lemma 1.9. Thus, the computation path of  $u$  from  $p$  to  $q$  must leave  $B_i$  and reenter through  $s_i$ . By the same reason, the computation path of  $u^{r-1}$  from  $q$  to  $p$  must leave  $B_i$  and reenter through  $s_i$ . First consider the case  $i$  is even. Let  $v$  be the shortest suffix of  $u$  such that  $\delta'(s_i, v) = q$ . Let  $w$  be the shortest suffix of  $u^{r-1}$  such that  $\delta'(s_i, w) = p$ . Then  $|v| = |w|$  and  $q = p$ , which is a contradiction. If  $i$  is odd, we can select the shortest suffix which starts at  $t_i$  instead, and argue as before.  $\square$

Let  $\text{DELTA} = (j - i) \bmod \text{PRIME}$ ,  $0 < \text{DELTA} < \text{PRIME}$ . Let  $p_0 = p$  and  $p_{k+1} = \delta'(p_k, u)$  for  $0 \leq k \leq r - 1$ . Thus,  $q = p_1$  and  $p = p_r$ . Let  $p_k \in Q_{i_k} - \{r_{i_k}\}$  for  $0 \leq k \leq r$ .

**Claim 1.13.** For all  $0 \leq k \leq r-1$ ,  $\text{DELTA} = (i_{k+1} - i_k) \bmod \text{PRIME}$ .

**Proof of Claim 1.13.** Actually, we have that  $\text{DELTA} = (\text{the number of substring } \langle \#, 0 \rangle \langle \#, 1 \rangle \langle \#, 2 \rangle \text{ in } u) \bmod \text{PRIME}$ , and by  $\langle \#, 0 \rangle \langle \#, 1 \rangle \langle \#, 2 \rangle$  we can move in  $B$  from  $B_i$  to  $B_{i+1}$ .  $\square$

**Claim 1.14.**  $r = \text{PRIME}$ .

**Proof of Claim 1.14.** If  $r < \text{PRIME}$ , then  $r \times \text{DELTA} \bmod \text{PRIME} \neq 0$ . Thus  $i_0 \neq i_k$ , i.e.  $p = p_0 \neq p_k$  for all  $1 \leq k < \text{PRIME}$ . Since  $r$  is the least number satisfying the condition, we conclude that  $r = \text{PRIME}$ .  $\square$

**Proof of Lemma 1.11 (conclusion).** Observe that  $\text{DELTA}$  is a generator of the cyclic group  $Z_{\text{PRIME}}$ . Therefore, the sequence  $i_0, i_1, \dots, i_{\text{PRIME}-1}$  is a cyclic permutation of  $1, 2, \dots, \text{PRIME}$ . Now, let  $u_1$  be the shortest prefix of  $u$  such that  $\delta'(p_k, u_1) = s_{i_k+1}$ , and  $u_3$  be the shortest suffix of  $u$  such that  $\delta'(s_{i_k+1}, u_3) = p_{k+1}$

$$\rightarrow s_{i_k} - u_3 \rightarrow p_k - u_1 \rightarrow s_{i_k+1} - u_2 \rightarrow s_{i_k+1} - u_3 \rightarrow p_{k+1} - u_1 \rightarrow s_{i_k+1+1} \rightarrow$$

Then  $u = u_1 u_2 u_3$  for some  $u_2$  and it holds that  $\delta'(s_{i_k+1}, u_2) = s_{i_k+1}$ . Consider  $u_3 u_1$ . Clearly,  $\delta'(s_i, u_3 u_1) = s_{i+1}$  for all  $i = 1, \dots, \text{PRIME} - 1$ . Further,  $\delta'(s_{\text{PRIME}}, u_3 u_1) = s_1$ . Let  $x$  be such that  $u_3 u_1 = x \langle \#, 2 \rangle$ . Then,  $x$  is accepted by all  $B_i$ 's. Let  $x = \langle a_1, 0 \rangle \langle a_2, 1 \rangle \dots \langle a_m, 2p(n)+1 \rangle \langle \#, 0 \rangle \langle \#, 1 \rangle$  and define  $y = a_1 a_2 \dots a_m \# \#$ . Then, clearly,  $y$  is accepted by all  $A_i$ 's. This completes the proof of Lemma 1.11.  $\square$

The remaining problem is that we have a variable-size input alphabet instead of a fixed-size alphabet. The idea is to encode such input symbols by binary strings of the same length that depends on the size of the input alphabet. Let  $B = (Q, \Sigma', \delta', s, \{f\})$  be the DFA constructed above. We construct a DFA  $B' = (Q', \{0, 1\}, \delta'', s', \{f'\})$  as follows:

$$Q' = Q \times \{0, 1\}^{\leq k-1}, \text{ where } k = \lceil \log_2 |\Sigma'| \rceil,$$

$$s' = \langle s, \varepsilon \rangle, \quad f' = \langle f, \varepsilon \rangle$$

and  $\delta''$  is so defined that

$$\delta''(\langle q, \varepsilon \rangle, x_a) = \langle p, \varepsilon \rangle,$$

where  $\delta'(q, a) = p$  and  $x_a = a_1 a_2 \dots a_k \in \{0, 1\}^*$  is a binary string which encodes the symbol  $a \in \Sigma'$ . Therefore, all intermediate transitions are defined as follows:

$$\delta''(\langle q, \varepsilon \rangle, a_1) = \langle q, a_1 \rangle,$$

$$\delta''(\langle q, a_1 \rangle, a_2) = \langle q, a_1 a_2 \rangle,$$

...

$$\delta''(\langle q, a_1 \dots a_{k-2} \rangle, a_{k-1}) = \langle q, a_1 \dots a_{k-1} \rangle,$$

$$\delta''(\langle q, a_1 \dots a_{k-1} \rangle, a_k) = \langle p, \varepsilon \rangle.$$

For all  $(\langle q, x \rangle, b)$  not defined by the above rule, we set

$$\delta''(\langle q, x \rangle, b) = \langle d, \varepsilon \rangle,$$

where  $d$  is the dead state of  $B'$ . Now, we can easily see that if there is a nontrivial cycle within  $B$ , then there is a nontrivial cycle within  $B'$ . However, the converse is not necessarily true, as shown in the following example.

**Example 1.15.** Let us consider a cycle  $q_0 a_1 q_5 a_2 q_{10} a_3 q_0$  in the DFA in Example 1.6, where all  $q_i$ 's and  $a_j$ 's are distinct. Let us encode  $a_1$  as 00100,  $a_2$  as 10010,  $a_3$  as 01001. The cycle in the modified DFA becomes:  $q_0 0 q_1 0 q_2 1 q_3 0 q_4 0 q_5 1 q_6 0 q_7 0 q_8 1 q_9 0 q_{10} 0 q_{11} 1 q_{12} 0 q_{13} 0 q_{14} 1 q_0$ , where all  $q_i$ 's are distinct, and all  $q_i$ 's except  $q_0$ ,  $q_5$  and  $q_{10}$  are intermediate states. Obviously,  $q_0$ , 001 and  $(001)^5$  define a nontrivial cycle in the modified DFA, whereas there is no nontrivial cycle in the original DFA.

We need the following encoding schema to avoid the above possibility. We encode 0 by 01 and 1 by 10. Thus, 00100 becomes  $y = 0101100101$ . We also concatenate  $x = 11111111100$  with  $y$  which gives  $xy = 111111111000101100101$ , where the number of 1's in  $x$  is equal to  $|y|$ . Let  $q, u, r$  define a nontrivial cycle in the modified DFA. Then,  $|u|$  must be a multiple of  $|xy|$ , and  $u$  can be written as  $u = vwx$  such that  $w$  is a concatenation of encodings of  $a_i$ 's. Then  $u' = wxv$  is a concatenation of encodings of  $a_i$ 's and  $\delta(q, v)$ ,  $u'$  and  $r$  defines a nontrivial cycle. Now it is not hard to see that if there is a nontrivial cycle in the modified DFA, then there is a nontrivial cycle in the original DFA.

We apply the above idea as follows. The form of the encodings of  $a_i$ 's is  $1^{2k}00\{01, 10\}^k$ , where  $k$  is the length of the binary encodings of symbols in the original DFA  $B'$ . By an argument similar to the one in Example 1.12, the length of  $u$  must be a multiple of  $4k + 2$ , and for  $u$  there is  $u'$  which is a concatenation of encodings of  $a_i$ 's and defines a nontrivial cycle. Thus, if there is a nontrivial cycle in  $B'$ , then there is a nontrivial cycle in  $B$ .

Thus, we have proved the following lemma.

**Lemma 1.16.** *Finite-automaton cycle existence with input alphabet  $\Sigma = \{0, 1\}$  is logspace-complete for PSPACE.*

From Lemma 1.16, we obtain the following theorem as a corollary.

**Theorem 1.17.** *Finite-automaton aperiodicity is logspace-complete for PSPACE.*

## 2. The complexity of dot-depth-one language recognition and piecewise testable language recognition

In this section we characterize the complexity of two other problems; namely, dot-depth-one language recognition and piecewise testable language recognition. We

show that these two problems are logspace-complete for NL, where NL is the class of languages accepted by nondeterministic logspace-bounded Turing machines. The following result is used.

**Proposition 2.1.** (Immerman [4]). *NL is closed under complement.*

We now introduce a condition which characterizes piecewise testable languages.

**Definition 2.2.** Given a DFA  $M = (Q, \Sigma, \delta, q_0, F)$ , we denote its transition diagram by  $G(M)$ . We also define  $G(M, \Gamma)$  by considering only transitions labeled by symbols in  $\Gamma$ , where  $\Gamma \subseteq \Sigma$ . Let  $p$  be a vertex of  $G$ . The component defined by  $p$ , written  $C(p)$ , is

$$C(p) = \{p\} \cup \{q \mid \text{there is a path from } p \text{ to } q\}.$$

**Proposition 2.3(a)** (Simon [8]). *Let  $W$  be a regular language and  $M$  be the minimum-state DFA accepting  $W$ .  $W$  is piecewise testable iff (1)  $G(M)$  is acyclic and (2) for any subset  $\Gamma$  of  $\Sigma$ , each component of  $G(M, \Gamma)$  has a unique maximal state, where a state is said to be maximal if from that state there is no outgoing transition labeled by  $\Gamma$ .*

Thus,  $W$  is not piecewise testable iff either (1)  $G(M)$  is cyclic or (2) there is one component of  $G(M, \Gamma)$  having two distinct maximal states.

**Observation** (Stern [9]). *If  $G(M)$  is acyclic, then  $q$  is a maximal state of a component  $C$  of  $G(M, \Gamma)$  iff (1)  $q \in C$  and (2)  $\Gamma \subseteq \Sigma(q) = \{a \in \Sigma \mid \delta(q, a) = q\}$ .*

From the above observation 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, Proposition 2.3(a) can be restated as follows.

**Proposition 2.3(b).**  *$W$  is not piecewise testable iff either (1)  $G(M)$  is cyclic or (2) there are 3 distinct states  $p, q, q'$  so that there are paths from  $p$  to  $q$  and  $p$  to  $q'$  in the graph  $G(M, \Sigma(q) \cap \Sigma(q'))$ .*

From Proposition 2.3(b) we have the following NL-algorithm which solves the piecewise testable language recognition problem.

**Lemma 2.4.** *Piecewise testable language recognition is in NL.*

**Proof.** Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a minimum state DFA.

(1) **if** there is a cycle in  $G(M)$  **then** return ('yes');

(2) guess  $p, q, q'$ ;

$s_1 := p; \quad s_2 := p;$

```

repeat guess  $a, b \in \Sigma(q) \cap \Sigma(q')$ ;
     $s_1 := \delta(s_1, a)$ ;
     $s_2 := \delta(s_2, b)$ ;
until  $s_1 = q$  and  $s_2 = q'$ ;
return ('yes');

```

Obviously, the above algorithm is in NL and gives a positive answer when  $M$  does not accept a piecewise testable language. Since NL is closed under complement [4], piecewise testable language recognition is in NL.  $\square$

**Lemma 2.5.** *Piecewise testable language recognition is NL-hard.*

**Proof.** We reduce graph accessibility (GAP for short), a well-known NL-complete problem, to piecewise testable language recognition. A special case of GAP is monotone 2GAP where out-degree of each vertex is bounded by 2 and for all edges  $e = \langle u, v \rangle$ ,  $v$  is greater than  $u$  (the vertices are linearly ordered). It is not hard to see that monotone 2GAP is also logspace-complete for NL. Let  $(G, s, g)$  be an instance of monotone 2GAP, where  $G = (V, E)$ ,  $V = \{1, 2, \dots, n\}$ ,  $s = 1$  and  $g = n$ .

We construct a minimum-state DFA  $M = (Q, \Sigma, \delta, p_1, \{f\})$ , where  $Q = V \cup \{f\} \cup \{p_i \mid 1 \leq i \leq n\} \cup \{q_i \mid 1 \leq i \leq n\}$ ,  $\Sigma = \{0, 1, 2\}$  and  $\delta$  is defined as follows (see Fig. 5):

$$\begin{aligned} \delta(p_i, 2) &= i \text{ for } 1 \leq i \leq n, \\ \delta(p_i, a) &= p_{i+1} \text{ for all } a \in \{0, 1\} \text{ and } 1 \leq i \leq n-1, \\ \delta(p_n, a) &= n \text{ for } a \in \{0, 1\}. \end{aligned}$$

For all  $i \in V - \{n\}$  we have the following cases:

outdegree( $i$ ) = 2: let  $j, k$  ( $j < k$ ) be two vertices adjacent to  $i$ . In this case

$$\delta(i, 0) = j, \quad \delta(i, 1) = k, \quad \delta(i, 2) = q_i;$$

outdegree( $i$ ) = 1: let  $j$  be the vertex adjacent to  $i$ . In this case

$$\delta(i, a) = j \text{ for } a \in \{0, 1\}, \quad \delta(i, 2) = q_i;$$

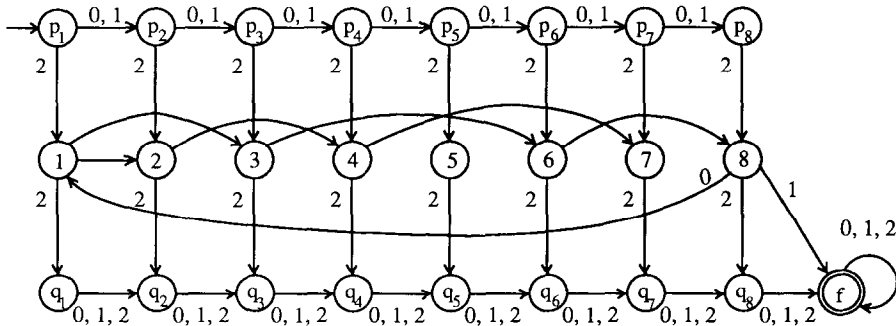


Fig. 5

$\text{outdegree}(i)=0$ :

$$\begin{aligned} \delta(i, a) &= f \text{ for } a \in \{0, 1\}, & \delta(i, 2) &= q_i, \\ \delta(n, 0) &= 1, \quad \delta(n, 1) = f, & \delta(n, 2) &= q_n. \end{aligned}$$

For all  $a \in \Sigma$  and  $1 \leq i \leq n-1$

$$\delta(q_i, a) = q_{i+1}, \quad \delta(q_n, a) = f, \quad \delta(f, a) = f.$$

(Note that the states  $p_i$ 's and  $q_i$ 's are introduced in order to obtain the minimality of the resulting DFA.)

Observe that if there is a path from  $s$  to  $g$ , then there is a cycle in  $G(M)$ . Further,  $f$  is the only state  $q$  such that  $\Sigma(q) \neq \emptyset$ . Therefore, if we can show that  $M$  is a minimum-state DFA, then we can conclude that  $(G, s, g)$  belongs to monotone 2GAP iff  $L(M)$  is not a piecewise testable language.

**Claim.**  $M$  is a minimum-state DFA.

**Proof of Claim.** First, observe that all  $p_i$ 's are pairwise inequivalent since if  $i < j$ , then  $\delta(p_i, 0^{n-j}2^3) \neq f$  and  $\delta(p_j, 0^{n-j}2^3) = f$ . Next, one can easily see that all  $q_i$ 's are pairwise inequivalent by a similar argument. Also, all states  $i = 1, \dots, n$  are pairwise inequivalent since if  $i < j$ , then  $\delta(i, 20^{n-j+1}) \neq f$  and  $\delta(j, 20^{n-j+1}) = f$ .

Note that for all  $1 \leq i, j \leq n$   $\delta(p_i, 0^{n-i+2}20^{n-1}) \neq f$ , but  $\delta(q_j, 0^{n-i+2}20^{n-1}) = f$ . Thus, all pairs  $p_i$ 's and  $q_j$ 's are pairwise inequivalent. Also, all the pairs  $p_i$ 's and  $j$ 's are pairwise inequivalent since if  $i \geq j+1$ , then  $\delta(p_i, 21^{n-j}) = f$ , but  $\delta(j, 21^{n-j}) \neq f$ ; if  $i < j+1$ , then  $\delta(j, 2^{n-j+2}) = f$ , but  $\delta(p_i, 2^{n-j+2}) \neq f$ . By a similar argument, all the pairs  $q_i$ 's and  $j$ 's are pairwise inequivalent. Thus, we conclude that  $M$  is minimal.  $\square$

**Proof of Lemma 2.5 (conclusion).** From the above claim, it follows that  $M$  does not recognize a piecewise testable language iff there is a path from  $s$  to  $g$  in  $G$ . The above reduction can be easily carried out by a deterministic logspace-bounded Turing machine. This completes the proof of Lemma 2.5.  $\square$

From Lemmas 2.4 and 2.5, we obtain the following theorem.

**Theorem 2.6.** *Piecewise testable language recognition is logspace-complete for NL.*

Next we introduce a condition that characterizes the dot-depth-one languages.

**Definition 2.7.** Let  $k$  be an integer. A DFA  $M$  is  $k$ -stable if for any two states  $p, q$  and any word  $w$  of length  $k$ , whenever  $p, q, \delta(p, w), \delta(q, w)$  belong to the same strongly connected component, then  $\delta(p, w) = \delta(q, w)$ .

Thus, a DFA  $M$  is not  $k$ -stable if there are states  $p, q$  and a word  $w$  of length  $k$  such that  $p, q, \delta(p, w)$  and  $\delta(q, w)$  belong to the same strongly connected component and  $\delta(p, w) \neq \delta(q, w)$ .

**Definition 2.8.** Two words  $u, v$  are  $k$ -coinitial if they have the same first  $k$  letters; we write  $c_k(u, v)$  if  $u$  and  $v$  are  $k$ -coinitial.

A *fork(k) of type I* is a diagram of the form described by Fig. 6, where  $u, v$  are  $k$ -coinitial words and  $A, A'$  are two distinct strongly connected components.

A *fork(k) of type II* is defined as in Fig. 7, with  $c_k(u, x), c_k(v, y)$  and  $A \neq A'$  are two distinct strongly connected components.

**Proposition 2.9** (Stern [9]). *A regular language is of dot-depth one iff for some  $k$ , its minimum-state DFA  $M$  is  $k$ -stable and admits no fork(k) of type I and type II. Further,  $k$  can be taken to be  $|Q|^3$ , where  $Q$  is the set of states of  $M$ .*

Thus, a regular language is not of dot-depth one iff its minimum state DFA  $M$  is not  $k$ -stable or admits fork(k) of type I or type II, where  $k = |Q|^3$  and  $Q$  is the set of states of  $M$ . From Proposition 2.9, we have the following NL algorithm for dot-depth-one language recognition.

**Lemma 2.10.** *Dot-depth-one language recognition is in NL.*

**Proof.** Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a minimum-state DFA.

(1) /\* Test whether  $M$  is not  $k$ -stable for  $k = |Q|^3$  \*/

guess  $p, q$ ;

if  $p, q$  belong to the same strongly connected component then

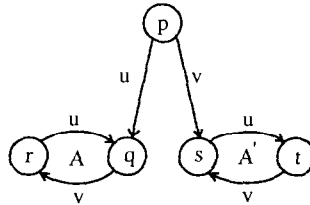


Fig. 6

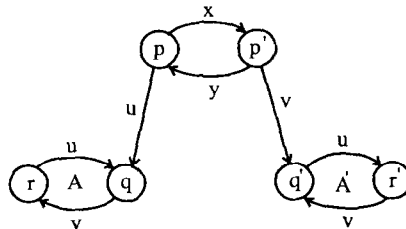


Fig. 7

```

begin
   $s_1 := p;$      $s_2 := q;$ 
  for  $i := 1$  to  $k$  do
    begin
      guess  $a \in \Sigma;$ 
       $s_1 := \delta(s_1, a);$ 
       $s_2 := \delta(s_2, a);$ 
    end;
    if  $s_1 \neq s_2$  then return ('yes');
  end
  (2) /* Test whether  $M$  admits fork( $k$ ) of type I */
  guess  $p, q, r, s, t;$ 
  if  $q, r$  and  $s, t$  constitute two different strongly connected components then
    begin
       $s_1 := p;$      $s_2 := r;$      $s_3 := s;$ 
      for  $i := 1$  to  $k$  do
        begin
          guess  $a \in \Sigma;$ 
           $s_1 := \delta(s_1, a);$ 
           $s_2 := \delta(s_2, a);$ 
           $s_3 := \delta(s_3, a);$ 
        end;
        if  $s_1 = q$  and  $s_2 = q$  and  $s_3 = t$  then
          begin
             $s_1 := p;$      $s_2 := t;$      $s_3 := q;$ 
            for  $i := 1$  to  $k$  do
              begin
                guess  $a \in \Sigma;$ 
                 $s_1 := \delta(s_1, a);$ 
                 $s_2 := \delta(s_2, a);$ 
                 $s_3 := \delta(s_3, a);$ 
              end;
              if  $s_1 = s$  and  $s_2 = s$  and  $s_3 = r$  then
                return ('yes');
            end
          end
        end
      (3) /* Test whether  $M$  admits fork ( $k$ ) of type II */
      Similar to that of fork ( $k$ ) of type I.
    end
  end

```

Note that computing strongly connected components and checking that the connected components are distinct are both in NL. Therefore, the above algorithm is in NL and gives a positive answer when  $M$  does not accept a dot-depth-one language. Since NL is closed under complement [4], dot-depth-one language recognition is in NL.  $\square$



**Lemma 2.11.** *Dot-depth-one language recognition is NL-hard.*

**Proof.** The proof is essentially similar to that of Lemma 2.5. We reduce monotone 2GAP to dot-depth-one language recognition. Actually, we reduce monotone 2GAP to the problem of checking whether a given minimum-state DFA  $M$  is  $k$ -stable or not. The details are left to the reader as an exercise.  $\square$

From Lemmas 2.10 and 2.11 we obtain the following theorem.

**Theorem 2.12.** *Dot-depth-one language recognition is logspace-complete for NL.*

### 3. Conclusions

In this paper we have characterized the exact complexity of three problems: (1) finite-automaton aperiodicity, (2) dot-depth-one language recognition and (3) piecewise testable language recognition. For all the three problems, the DFAs in the input are assumed to be minimum-state DFAs. Since testing whether a given DFA is minimal is known to be in P, finite-automaton aperiodicity remains PSPACE-complete even without the minimality assumption. In [1] we showed that minimization of DFAs is NL-complete. Therefore, dot-depth-one language recognition and piecewise testable language recognition remain NL-complete even when the DFAs in the input are not assumed to be minimal.

### Acknowledgment

The authors thank an anonymous referee for some helpful remarks that improve the presentation of this paper.

### References

- [1] S. Cho and D.T. Huynh, The parallel complexity of finite state automata problems, Tech. Report UTDCS-22-88, Univ. of Texas, Dallas, 1988.
- [2] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers* (Oxford University Press, London, 3rd ed., 1954) 343.
- [3] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
- [4] N. Immerman, Nondeterministic space is closed under complement, *SIAM J. Comput.* **17** (1988) 935–938.
- [5] D. Kozen, Lower bounds for natural proof systems, in: *Proc. 18th IEEE Ann. Symp. on Foundations of Comput. Sci.* (1977) 254–266.

- [6] W.J. Savitch, Relationship between nondeterministic and deterministic tape complexities, *J. Comput. System. Sci.* **4** (1970) 177–192.
- [7] M.P. Schützenberger, On finite monoids having only trivial subgroups, *Inform. and Control* **8** (1965) 190–194.
- [8] I. Simon, Piecewise testable events, *Lecture Notes in Computer Science*, Vol. **33** (Springer, Berlin, 1975) 214–222.
- [9] J. Stern, Complexity of some problems from the theory of automata, *Inform. and Control* **66** (1985) 163–176.