# A Polynomial Time Algorithm for the Local Testability Problem of Deterministic Finite Automata

Sam M. Kim, *Member, IEEE,* Robert McNaughton, and Robert McCloskey

*Abstract*—**We investigate the local testability problem of deterministic finite automata. A locally testable language is a language with the property that, for some nonnegative integer $k$, whether or not a word $w$ is in the language depends on 1) the prefix and suffix of $w$ of length $k$, and 2) the set of substrings of $w$ of length $k+1$, without regard to the order in which these substrings occur. The local testability problem is, given a deterministic finite automaton, to decide whether or not it accepts a locally testable language. Until now, no polynomial time algorithm for this problem has appeared in the literature. We present an $O(n^2)$ time algorithm for the local testability problem based on two simple properties that characterize locally testable automata.**

*Index Terms*— **Algorithms, finite automata, local testability, regular languages, state transition graphs.**

## I. Introduction

LOCAL testability, a concept in the study of pattern recognition, is best understood in terms of a kind of computational procedure used to classify a two-dimensional image: a window of relatively small size is moved around on the image and a record is made of the various attributes of the image that are detected by what is observed through the window. No record is kept of the order in which the attributes are observed, where each attribute occurs, or how many times it occurs. We say that a class of images is *locally testable* if a decision about whether a given image belongs to the class can be made simply on the basis of the set of attributes that occur. In [12], local testability is discussed in terms of "diameter-limited perceptrons."

In this paper we are concerned with the local testability of one-dimensional images. We shall think of such an image as a character string, and the classification as a language. More technically, we define a locally testable language to be a language with the property that for some nonnegative integer $k$, whether or not a word $w$ is in the language depends on 1) the prefix and the suffix of $w$ of length $k$, and 2) the set of substrings of $w$ of length $k+1$, without regard to the order in which these substrings occur or the number of times each substring occurs. It is easy to show that locally testable languages are regular. Locally testable languages are a generalization of the definite and reverse-definite languages, which can be found, for example, in [4] and [13]. Since locally testable languages were first introduced in 1971 [10], they have been extensively investigated [2], [6], [9], [18].

This paper investigates the local testability problem of deterministic finite automata and presents a polynomial time algorithm. The local testability problem is, given a deterministic finite automaton, to determine whether the automaton accepts a locally testable language. As far as we know, all previous algorithms for this problem take exponential time [2], [9]. If the language is given not as a deterministic automaton but as a semigroup (i.e., the syntactic semigroup of the language) the algorithm is quite simple; the time complexity is linear in the size of the multiplication table of the semigroup. But when the language is given as a deterministic finite automaton, using the syntactic semigroup does not give us a polynomial time algorithm because its size is generally exponential in the number of states of the automaton.

Although our polynomial time algorithm is new, various things were known about testing a language for local testability and about other similar problems. Hunt and Rosenkrantz [6] proved that the problem of deciding whether a regular language is locally testable is PSPACE-hard when the language is given as a regular expression. Two recent papers [16], [17] offer an interesting characterization of the class of piecewise testable languages and give a polynomial time algorithm for deciding whether the language of a deterministic finite automaton is in this class.

This paper proves that a reduced deterministic finite automaton is locally testable if and only if its state transition graph has two simple properties. The first is a property of each maximal strongly connected component, and the second a relationship between pairs of these components. This new simple characterization of state transition graphs of locally testable automata leads us to an efficient polynomial time algorithm for the local testability problem. The algorithm takes $O(sn^2)$ time to see if the graph has the two properties, where $s$ is the alphabet size and $n$ is the number of states of the automaton.

This paper has four sections including this one. Section II introduces definitions and notation for the basic properties of state transition graphs. Section III gives the characterization of locally testable automata and Section IV presents a polynomial time algorithm for the local testability problem. The paper closes with a discussion of open problems and suggestions for future research.

## II. Notation and Terminiology

We assume that a problem instance is given in terms of

Manuscript received November 23, 1989; revised August 12, 1990. This work was supported in part by the Directorate of Computer and Information Science and Engineering of the National Science Foundation under Institutional Infrastructure Grant CDA-8805910.

The authors are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180.

IEEE Log Number 9102589.

the state transition graph of a reduced deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$. Our notation follows [5]. For simplicity, we will omit from our illustrations the nonaccepting sink state, which has transitions to itself on all input symbols. We shall use the term *connected component* (sometimes just *component*) to refer to any subgraph (of a transition graph) whose underlying undirected graph is connected. By SCC we mean a maximal strongly connected component. We say that the SCC $m_1$ is an ancestor of the SCC $m_2$ (equivalently, $m_2$ is a descendant of $m_1$) if there is a path from $m_1$ to $m_2$.

*Definition 1 (substring vector):* Let $\Sigma$ be a finite alphabet and $x \in \Sigma^*$. For an integer $k \geq 0$, define $f_k(x)$, $t_k(x)$, and $I_{k+1}(x)$ as follows. For $|x| \leq k$, $f_k(x) = t_k(x) = x$ and $I_{k+1}(x) = \emptyset$. For $|x| > k$, $f_k(x)$ and $t_k(x)$ are, respectively, the prefix and suffix of $x$ of length $k$; $I_{k+1}(x)$ is the set of substrings of $x$ of length $k+1$, i.e., $\{v | x = uvw$, for some $u, w \in \Sigma^*$, and $|v| = k + 1\}$. The substring vector of order $k + 1$ of $x$ is the ordered triple $\mathrm{SV}_{k+1}(x) = (f_k(x), I_{k+1}(x), t_k(x))$.

*Definition 2 (locally testable languages)* [2], [10]; Let $L \subseteq \Sigma^*$.

a) $L$ is 0-testable iff it is $\Sigma^*$ or $\emptyset$.
b) For an integer $k \geq 0$, $L$ is $(k+1)$-testable iff, for all $x$, $y \in \Sigma^*$, $\mathrm{SV}_{k+1}(x) = \mathrm{SV}_{k+1}(y)$ implies that either both $x$ and $y$ are in $L$ or neither is in $L$.
c) $L$ is locally testable iff it is $k$-testable for some $k \geq 0$.

These definitions are in accord with [2]. The definition of $k$-testability is different in [10] although it gives rise to the same class of locally testable languages, as is pointed out in [2].

*Definition 3 (locally testable automaton):* A finite automaton is locally testable iff the language it accepts is locally testable. The local testability problem of deterministic finite automata is that of deciding whether a given deterministic automaton is locally testable. Fig. 1 shows examples.

Automaton $M_1$ is 3 testable. (Indeed, the language of $M_1$ consists of the string $b$ and the strings of length 2 or more that 1) do not begin with $ba$, 2) have no substring $aba$, and 3) end in $ab$.) $M_2$ is not locally testable. (For any $k \geq 0$, $a^{2k+1}$ is in the language of $M_2$ but $a^{2k+2}$ is not. Since $f_k(a^{2k+1}) = f_k(a^{2k+2}) = a^k$, $t_k(a^{2k+1}) = t_k(a^{2k+2}) = a^k$, and $I_{k+1}(a^{2k+1}) = I_{k+1}(a^{2k+2}) = \{a^{k+1}\}$, these two strings have the same substring vector of order $k+1$, and so this language is not $(k+1)$-testable. Since it is obviously not 0-testable either, it follows that the automaton is not locally testable.)

*Definition 4 (transition span):* Let $M = (Q, \Sigma, \delta, q_0, F)$ be an automaton, and let $m$ be a connected component of the state transition graph of $M$. The *transition span* in $m$ of a state $p \in Q$ is the set $\mathrm{TS}(m, p) = \{x | x \in \Sigma^*$, for every prefix $w$ of $x$, $\delta(p, w)$ is in $m\}$. Notice that if $p$ is not in $m$, $\mathrm{TS}(m, p)$ is empty.

*Definition 5 (TS-equivalence):* Let $m$ be a connected component of the state transition graph of an automaton. States $p$ and $q$ are TS-equivalent in $m$ iff $\mathrm{TS}(m, p) = \mathrm{TS}(m, q)$. Clearly, TS-equivalence is an equivalence relation. Also, if $p$ and $q$ are TS-equivalent in $m$, then, for any $w \in \Sigma^*$, so are $r$ and $s$, where $r = \delta(p, w)$ and $s = \delta(q, w)$.

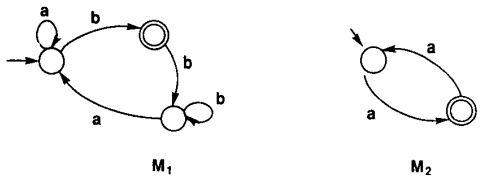



Fig. 1. $M_1$: locally testable; $M_2$: not locally testable.

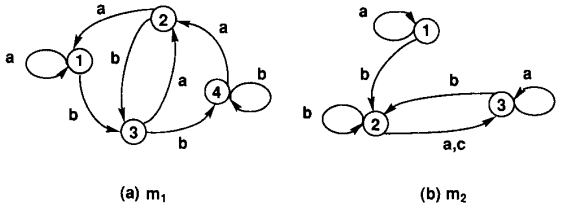



Fig. 2. Two illustrations of TS-equivalence.

For example, in Fig. 2(a), every pair of states $i, j \in \{1, 2, 3, 4\}$ are TS-equivalent in $m_1$ because $\mathrm{TS}(m_1, i) = \mathrm{TS}(m_1, j) = \{a, b\}^*$. In Fig. 2(b), states 1 and 3 are TS-equivalent in $m_2$, since $\mathrm{TS}(m_2, 1) = \mathrm{TS}(m_2, 3)$. On the other hand, states 1 and 2 are not TS-equivalent, since $\delta(1, c)$ is not in $m_2$ while $\delta(2, c) = 3$ is.

*Definition 6 (reaching component).* Let $m_i$ and $m_j$ be two components of a state transition graph. The *reaching component* from $m_i$ to $m_j$, written $m_{ij}$, is the subgraph which comprises $m_i$ and $m_j$ together with all directed paths which start in $m_i$ and end in $m_j$. If $m_i$ is a single state, say $p$, then the reaching component from $p$ to $m_j$ is denoted by $m_{pj}$.

*Definition 7 (pair-graph)* [11]: Let $M = (Q, \Sigma, \delta, q_0, F)$ be an automaton. Let $Q_1$ and $Q_2$ be two subsets of $Q$; these need not be distinct. Let * be a symbol not in $Q$. The pair-graph on $Q_1 \times Q_2$ is the edge-labeled directed graph $G(V, E)$, where $V = (Q_1 \cup \{*\}) \times (Q_2 \cup \{*\}) - \{(*, *)\}$ and $E$ is defined as follows (see Fig. 3, for an example):

Define $\delta_i$: $Q_i \times \Sigma \rightarrow Q_i \cup \{*\}$, $i = 1, 2$, such that, for all $p \in Q_i$ and $a \in \Sigma$,

$$\delta_i(p, a) = \begin{cases} q & \text{if } \delta(p, a) = q \in Q_i \\ * & \text{if } \delta(p, a) \text{ is not in } Q_i. \end{cases}$$

Then $E = \{((p, q), a, (r, s)) | p \in Q_1, q \in Q_2, p \neq q, r \in Q_1 \cup \{*\}, s \in Q_2 \cup \{*\}, (r, s) \neq (*, *), \delta_1(p, a) = r$, and $\delta_2(q, a) = s\}$. Notice that from a node $(p, *)$, $(*, q)$, or $(p, p)$ of the pair-graph, no outgoing edge exists. A pair-graph is not necessarily connected.

*Definition 8 (s-local, pairwise s-local)*: Let $M = (Q, \Sigma, \delta, q_0, F)$ be an automaton.

a) A component $m_1$ of the state transition graph is s-local (or strongly local) iff there is no pair of states $p$ and $q$ in $m_1$ and $x \in \Sigma^+$ such that $\delta(p, x) = p$ and $\delta(q, x) = q$.
b) Let $m_1$ and $m_2$ be two disjoint components of the state transition graph. Then $m_1$ and $m_2$ are pairwise s-local iff there is no pair of states $p$ and $q$, respectively, in $m_1$ and $m_2$ and $x \in \Sigma^+$ such that $\delta(p, x) = p$ and $\delta(q, x) = q$.

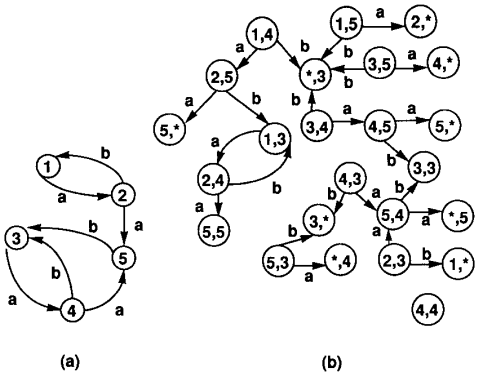



Fig. 3. (a) A component of a state transition graph. (b) The pair-graph on $Q_1 \times Q_2 = \{1,2,3,4,5\} \times (3,4,5)$.

In [8], a finite state machine is defined as having finite input memory span if the machine has no state pair $p$ and $q$ such that $\delta(p,x) = p$ and $\delta(q,x) = q$, for any nonnull input string $x$. The above definition is a generalization of this concept.

*Lemma 1:* Let $m_i$ and $m_j$ be distinct SCC's of the state transition graph of an automaton $M(Q,\Sigma,\delta,q_0,F)$. Let $Q_i$ and $Q_j$ be the sets of states, respectively, in $m_i$ and $m_j$.

a) The component $m_i$ is s-local iff the pair-graph on $Q_i \times Q_i$ has no cycle.
b) The components $m_i$ and $m_j$ are pairwise s-local iff the pair-graph on $Q_i \times Q_j$ has no cycle.

*Proof:* a) Suppose, for distinct states $p$, $q \in Q_i$ and $x \in \Sigma^+$, $\delta(p,x) = p$ and $\delta(q,x) = q$. For every prefix $x'$ of $x$, $\delta(p,x')$ and $\delta(q,x')$ are distinct states in $Q_i$, since $m_i$ is maximally strongly connected. Hence, the pair-graph on $Q_i \times Q_i$ has a cycle. Conversely, if $Q_i \times Q_i$ has a cycle, there must exist a pair of states $p$, $q \in Q_i$ and $x \in \Sigma^+$ such that $\delta(p,x) = p$ and $\delta(q,x) = q$. The proof of b) is similar. □

*Definition 9 (TS-local):* Let $m_j$ be an SCC of the state transition graph of a deterministic finite automaton. The graph is TS-local w.r.t. $m_j$ iff it has the following property. For every SCC $m_i$ which is an ancestor of $m_j$, either

a) $m_i$ and $m_j$ are pairwise s-local, or otherwise,
b) there exists a pair of states $p$ and $q$, in $m_i$ and $m_j$ such that $p$ and $q$ are TS-equivalent in $m_{ij}$ (the reaching component from $m_i$ to $m_j$).

The state transition graph of an automaton is TS-local iff, for every SCC $m_j$ of the graph, it is TS-local w.r.t. $m_j$.

*Lemma 2:* Let $M = (Q,\Sigma,\delta,q_0,F)$ be a deterministic automaton all of whose SCC's are s-local. For any $x \in \Sigma^*$, $p$, $q \in Q$, and $n \geq |Q|$, if $\delta(p,x^n) = q$, then $\delta(q,x) = q$.

*Proof:* Since $n \geq |Q|$, we have $\delta(p,x^h) = \delta(p,x^{h+j}) = q'$, for some $q' \in Q$, $h \geq 0$, $j \geq 1$, $h + j \leq n$. Thus, $\delta(q',x^j) = q'$. Let $\delta(q',x) = r$. Then $\delta(r,x^j) = r$. Since $q'$ and $r$ are in the same s-local SCC, by Definition 8a) it must be that $r = q'$ which is the $q$ of the statement of the lemma. □

The following lemma provides a simple way of finding a pair of states $p$ and $q$ which belong to SCC's $m_1$ and $m_2$, respectively, and are TS-equivalent in $m_{12}$, if there exists such a pair.

*Lemma 3:* Let $m_1$ and $m_2$ be SCC's such that $m_1$ is an ancestor of $m_2$ and $m_2$ is s-local. Let $r$ and $s$ be states, respectively, in $m_1$ and $m_2$ such that $\delta(r,x) = r$ and $\delta(s,x) = s$, for some $x \in \Sigma^+$. If there are states $p$ and $q$, respectively, in $m_1$ and $m_2$ that are TS-equivalent in $m_{12}$, then $r$ and $s$ are also TS-equivalent in $m_{12}$.

*Proof:* Find $w \in \Sigma^*$ such that $\delta(p,w) = r$. Then we have $\delta(p,wx^i) = r$, for all $i \geq 0$. Let $\delta(q,w) = q'$, $n = |Q|$, and $\delta(q',x^n) = s'$, i.e., $\delta(q,wx^n) = s'$. To complete the proof, it is enough to show that $s' = s$. (Note the remark at the end of Definition 5.) Since states $p$ and $q$ are TS-equivalent in $m_{12}$ and $\delta(p,wx^n) = r$ is in $m_1$, it follows that $\delta(q,wx^n) = s'$ is in $m_{12}$, and hence in $m_2$. By Lemma 2, the state $s' = \delta(q',x^n)$ satisfies $\delta(s',x) = s'$. Since $m_2$ is s-local, we have $s = s'$. □

We end this section by proving a lemma which leads to an efficient polynomial time algorithm in Section IV.

*Lemma 4:* Let $m_j$ be an s-local SCC of the state transition graph of a deterministic finite automaton $M = (Q,\Sigma,\delta,q_0,F)$, and let $m_o$ be the component of the graph which consists of all the states from which $m_j$ is reachable. (Notice that $m_o$ includes $m_j$.) Let $Q_o$ and $Q_j$ be the sets of states in $m_o$ and $m_j$, respectively, and let $G_{oj}$ be the pair-graph on $Q_o \times Q_j$. Then the state transition graph of $M$ is not TS-local w.r.t. $m_j$ iff there is a path in $G_{oj}$ from an SCC to a node of the form $(t,*)$ or $(*,t)$.

*Proof:* Assume that the transition graph of $M$ is not TS-local w.r.t. $m_j$. Then, for some SCC $m_i$ which is an ancestor of $m_j$, 1) $m_i$ and $m_j$ are not pairwise s-local, and 2) there is no pair of states, respectively, in $m_i$ and $m_j$ that are TS-equivalent in $m_{ij}$. By 1) and Lemma 1, there is an SCC in $G_{oj}$. Let $(p,q)$ be a node in that SCC. Condition 2) tells us that $p$ and $q$ are not TS-equivalent in $m_{ij}$. Hence, there exists $w \in \Sigma^+$ such that either $\delta(p,w)$ is in $m_{ij}$ and $\delta(q,w)$ is outside of $m_{ij}$, or else $\delta(p,w)$ is outside of $m_{ij}$, but $\delta(q,w)$ is in $m_{ij}$. Let $w$ be the shortest such word. Then, by the construction of the pair-graph, $G_{oj}$ must have a path from $(p,q)$ to a node of type either $(t,*)$ or $(*,t)$ with span $w$ for some state $t$ in $m_{ij}$.

Now, assume that from an SCC in $G_{oj}$ there is a path to a node of the form $(t,*)$ or $(*,t)$. Let $(p,q)$ be a node in the SCC, and let $x$ be the span of a cycle on $(p,q)$, i.e., $\delta(p,x) = p$ and $\delta(q,x) = q$. It follows that $p \neq q$ by the construction of the pair-graph. The state $p$ is not in $m_j$ since $q$ is in $m_j$ which is s-local. Since $\delta(p,x) = p$, the state $p$ must belong to an SCC, say $m_i$, which, by definition of $m_o$, must be an ancestor of $m_j$. Furthermore, Lemma 3 tells us that if $p$ and $q$ are not TS-equivalent in $m_{ij}$, then no two states of $m_i$ and $m_j$, respectively, are TS-equivalent in $m_{ij}$. Suppose first that there is a path in $G_{oj}$ from $(p,q)$ to $(t,*)$. Let $w$ be the span of this path. Then $\delta(p,w) = t$ is in $Q_o$ and $\delta(q,w)$ is not in $Q_j$. It follows that $\delta(p,w)$ is in $m_{ij}$ but $\delta(q,w)$ is not in $m_{ij}$ showing that $p$ and $q$ are not TS-equivalent in $m_{ij}$, and thus that $M$ is not TS-local w.r.t $m_j$. Similarly we can conclude that, if there is a path in $G_{oj}$ from $(p,q)$ to $(*,t)$, $M$ is not TS-local w.r.t. $m_j$. □

## III. Characterization of Locally Testable Deterministic Finite Automata

Using the lemmas in the preceding section, this section introduces a new characterization theorem which leads to an efficient polynomial time algorithm in Section IV. We start this section with the following known result, which will be used for the proof of our main theorem.

*Theorem 1* [2], [9]. A reduced automaton $M = (Q, \Sigma, \delta, q_0, F)$ is locally testable if and only if, for all $q \in Q$, $x \in \Sigma^+$, $y$, $z \in \Sigma^*$, and $n \geq |Q|$,

a) $\delta(q, x^n y x^n) = \delta(q, x^n y x^n y x^n)$, and
b) $\delta(q, x^n y x^n z x^n) = \delta(q, x^n z x^n y x^n)$.

For an arbitrary deterministic finite automaton, no practical algorithm that directly checks these properties has ever appeared in the literature. In [2] and [9], a characterization of locally testable automata was given in terms of an algebraic property of the semi-groups of the automata. As a computational method, this idea does not give us a polynomial time algorithm since the semigroup of a deterministic finite automaton is exponential in the size of its state transition graph in the worst case. We introduce the following simple characterization which can be examined in polynomial time.

*Theorem 2 (characterization):* A reduced deterministic finite automaton is locally testable iff it satisfies the following conditions:

a) All SCC's of the state transition graph are s-local.
b) The state transition graph is TS-local.

*Proof:* The proof of necessity is contained in the next two lemmas.

*Lemma 5:* If a finite state automaton $M$ is locally testable, then all SCC's in the reduced deterministic state transition graph of $M$ are s-local.

*Proof:* Let $Q$ be the set of states of the automaton, and suppose that there is an SCC $m_1$ in the state graph which is not s-local. Let $Q_1$ be the set of states in $m_1$. Then there are distinct states $p$, $q \in Q_1$ such that $\delta(p, x) = p$ and $\delta(q, x) = q$, for some $x \in \Sigma^+$. Pick words $y$, $z \in \Sigma^*$ such that $\delta(p, y) = q$ and $\delta(q, z) = p$. (See Fig. 4.) For any $n \geq |Q|$, we have $\delta(p, x^n y x^n z x^n) = p = \delta(p, x^n z x^n y x^n)$ by property b) of Theorem 1. Let $r = \delta(p, x^n z)$; then we get $\delta(r, x^n y x^n) = p \neq \delta(r, x^n y x^n y x^n) = q$, and property a) of Theorem 1 is violated. □

*Lemma 6:* Let $M$ be a reduced deterministic finite automaton which is locally testable. Then the transition graph of $M$ is TS-local.

*Proof:* Suppose that the transition graph is not TS-local. Then there exists a pair of SCC's $m_1$ and $m_2$ such that $m_1$ is an ancestor of $m_2$ and

a) $m_1$ and $m_2$ are not pairwise s-local, and
b) there is no pair of states $p$ and $q$, respectively, in $m_1$ and $m_2$ which are TS-equivalent in $m_{12}$.

Since $m_1$ and $m_2$ are not pairwise s-local, there exists a state pair $p$ and $q$, respectively, in $m_1$ and $m_2$, and $x \in \Sigma^+$ such that $\delta(p, x) = p$ and $\delta(q, x) = q$. By b) above, $p$ and $q$ are not TS-equivalent, which implies that there exists a word $u \in \Sigma^+$ satisfying either Case 1 or Case 2 below.

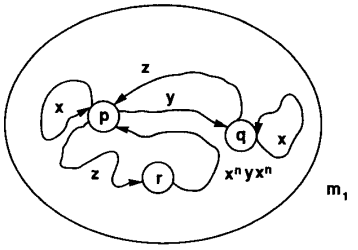



Fig. 4. An SCC which is not s-local.

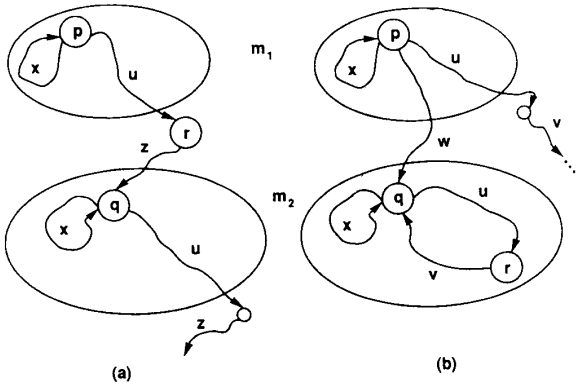



Fig. 5. Illustrations for the proof of Lemma 6.

*Case 1:* $\delta(p, u) = r$, for some state $r$ in $m_{12}$, while $\delta(q, u)$ is not in $m_{12}$. Since $m_2$ is maximally strongly connected, there can be no path from $\delta(q, u)$ back to $q$. [See Fig. 5(a).] Find $z \in \Sigma^*$ such that $\delta(r, z) = q$. Then we have $\delta(p, x^n u z x^n) = q \neq \delta(p, x^n u z x^n u z x^n)$, for any positive integer $n$, contradicting Theorem 1a).

*Case 2:* $\delta(q, u) = r$, for some state $r$ in $m_{12}$, while $\delta(p, u)$ is not in $m_{12}$. Note that state $r$ must be in $m_2$ because no state outside of $m_2$ which is reachable from $m_2$ belongs to $m_{12}$. [See Fig. 5(b).] Find $v$, $w \in \Sigma^*$ such that $\delta(p, w) = q$ and $\delta(r, v) = q$. Then, for any positive integer $n$, we have $\delta(p, x^n w x^n u v x^n) = q \neq \delta(p, x^n u v x^n w x^n)$, contradicting Theorem 1(b). □

This concludes the proof of the necessity of the two conditions of Theorem 2. The proof of sufficiency is contained in the next lemma.

*Lemma 7:* Let $M = (Q, \Sigma, \delta, q_0, F)$ be a reduced deterministic finite automaton whose transition graph satisfies conditions a) and b) of Theorem 2. Then, for every $p \in Q$, $x \in \Sigma^+$, $y$, $z \in \Sigma^*$, and $n \geq |Q|$,

1) $\delta(p, x^n y x^n) = \delta(p, x^n y x^n y x^n)$, and
2) $\delta(p, x^n y x^n z x^n) = \delta(p, x^n z x^n y x^n)$.

*Proof:* Suppose that the lemma does not hold. We consider the following two cases.

*Case 1:* There exists $p \in Q$ such that $\delta(p, x^n y x^n) = r \neq \delta(p, x^n y x^n y x^n) = s$ for some $r$, $s \in Q$, $x \in \Sigma^+$, $y \in \Sigma^*$. [See Fig. 6(a).]

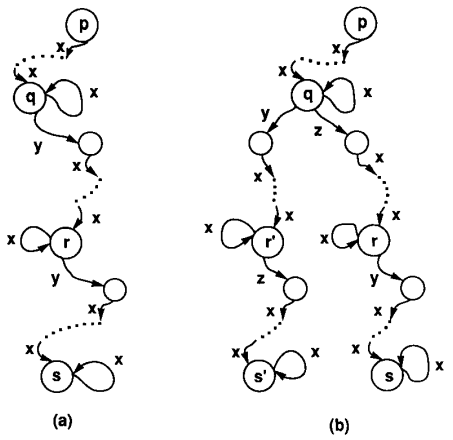



Fig. 6. Illustrations for the proof of Lemma 7.

Let $\delta(p, x^n) = q$; then $\delta(q, yx^n) = r$, and $\delta(r, yx^n) = s$. By Lemma 2, we have $\delta(q, x) = q$, $\delta(r, x) = r$, and $\delta(s, x) = s$. Since $r \neq s$, it follows that $q \neq r$. Since all SCC's are s-local, $q$, $r$, and $s$ are in three different SCC's. It follows that no two of those three SCC's are pairwise s-local.

Now, let $m_1$ and $m_2$ be the SCC's which include the states $q$ and $r$, respectively. Since the transition graph is TS-local, there exist two states, one in $m_1$ and the other in $m_2$, which are TS-equivalent in $m_{12}$. By Lemma 3, $q$ and $r$ are such a pair. However, they are not TS-equivalent in $m_{12}$ because $\delta(q, yx^n) = r$ is in $m_{12}$, while $\delta(r, yx^n) = s$ is not in $m_{12}$. We are in contradiction.

*Case 2:* There exists a state $p \in Q$ such that $\delta(p, x^n yx^n zx^n) = s' \neq \delta(p, x^n zx^n yx^n) = s$, for some $s$, $s' \in Q$, $x \in \Sigma^+$, $y$, $z \in \Sigma^*$, and $n \geq |Q|$. [See Fig. 6(b).] Let $\delta(p, x^n) = q$, $\delta(q, yx^n) = r'$ and $\delta(q, zx^n) = r$. Thus, $\delta(r', zx^n) = s'$ and $\delta(r, yx^n) = s$. By Lemma 2, we have $\delta(q, x) = q$, $\delta(r', x) = r'$, $\delta(r, x) = r$, $\delta(s', x) = s'$, and $\delta(s, x) = s$.

Let $m_q$, $m_r$, $m_s$, $m_{r'}$, and $m_{s'}$ be the SCC's that include the states $q$, $r$, $s$, $r'$, and $s'$, respectively. Since all SCC's are s-local, we know that if any two of these five states are distinct, then the two corresponding SCC's must be distinct. In particular, $m_s$ and $m_{s'}$ are distinct SCC's. Note that it cannot be that $q = r = r'$; for that would imply that $q = s$ and $q = s'$, contradicting $s \neq s'$. So either $q \neq r$ or $q \neq r'$. Without loss of generality, assume $q \neq r'$ and thus $m_q$ and $m_{r'}$ are distinct. We consider the following two cases.

*Case 2-1:* $q = r$. Then $r' = s$. Furthermore, $q \neq s$; otherwise, we would have $s = s'$. Since $s' \neq s = r'$, we know that $s'$ is not in $m_{r'}$. Thus, there is no path from $s'$ to $r'$, which implies that $s'$ is not in $m_{qr'}$. The two SCC's $m_q$ and $m_{r'}$ are not pairwise s-local; hence, by Lemma 3, $q$ and $r'$ must be TS-equivalent in $m_{qr'}$. But now we have a contradiction: $\delta(q, zx^n) = r = q$ is in $m_{qr'}$, but $\delta(r', zx^n) = s'$ is not.

*Case 2-2:* $q \neq r$. By Lemma 3, the states $q$ and $s$ are TS-equivalent in $m_{qs}$. Since the state $r = \delta(q, zx^n)$ is in $m_{qs}$, $\delta(s, zx^n) = t$ must be in $m_{qs}$ and hence in $m_s$. By Lemma 2, $\delta(t, x) = t$. Since $m_s$ is s-local, $t = s$, i.e., $\delta(s, zx^n) = s$. Also, the states $r$ and $s$ are TS-equivalent in $m_{rs}$. Hence, by similar reasoning, $\delta(s, yx^n) = s$. We thus get $\delta(s, yx^n zx^n) = s$. Since $q$ and $s$ are TS-equivalent in $m_{qs}$, $\delta(q, yx^n zx^n) = s'$ is in $m_{qs}$; this means there is a path from $s'$ to $m_s$, and hence a path from $s'$ to $s$.

By virtue of the conditions $q \neq r'$ and $q \neq r$, there is symmetry in Case 2-2: anything we can say about $y$, $z$, $w$, $r$, $r'$, $s$, and $s'$ is also true when we interchange $y$ with $z$, $r$ with $r'$, and $s$ with $s'$. Thus, the reasoning just above can be duplicated to show that there is a path from $s$ to $s'$. Thus, $s$ and $s'$ belong to the same SCC. Since all SCC's are s-local, we have $s = s'$, contradicting the assumption of Case 2 that $s \neq s'$. □

## IV. A Polynomial Time Algorithm for the Local Testability Problem

This section introduces an efficient polynomial time algorithm for the local testability problem of deterministic finite automata. The algorithm, given a state transition graph of a deterministic automaton, tests if the graph satisfies the conditions of Theorem 2, i.e.,

a) all SCC's of the state transition graph are s-local, and
b) the state transition graph is TS-local.

Let $G$ be the state transition graph of a deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, and let $s = |\Sigma|$ and $n = |Q|$. For each component $m_i$ of the state transition graph, let $Q_i$ denote the set of states in $m_i$. We assume $G$ is reduced. (If not, we can reduce it in $O(sn^2)$ time using a standard state reduction algorithm [5].) Testing condition a) of the theorem is easy. For each SCC $m_i$ construct the pair-graph on $Q_i \times Q_i$ and check if it is acyclic. Using a standard graph search algorithm [1], the test of condition a) can be done in $O(sn^2)$ time. It is not that easy to examine condition b) of the theorem, but Lemma 4 provides an idea. We describe an outline of the algorithm which examines $G$ for this condition.

Suppose that $G$ satisfies condition a) of Theorem 2. For an SCC $m_j$ in $G$, let $m_o$ be the component of the graph which consists of all the states from which $m_j$ is reachable. Let $Q_o$ and $Q_j$ be the sets of states in $m_o$ and $m_j$, respectively. The algorithm works as follows:

Repeat
1) Find an SCC $m_j$ on $G$ which has no descendant SCC;
2) Find $m_o$;
3) Construct the pair-graph $G_{oj}$ on $Q_o \times Q_j$.
4) Examine $G_{oj}$ and see if $G$ is TS-local w.r.t. $m_j$ using Lemma 4;
5) If $G$ is TS-local w.r.t. $m_j$ then $G := G - m_j$; (* delete $m_j$*) else exit (* $G$ is not locally testable *);

until ($G$ has no SCC) (* $G$ is TS-local *);

For the time complexity of this routine, suppose that $G$ has $r \geq 0$ SCC's and let $n_j$ be the number of states in SCC $m_j$, $1 \leq j \leq r$. Steps 1 and 2 can be done in $O(sn)$ time. Steps 3 and 4 together can be done in time $O(s|Q_o||Q_j|) \leq O(snn_j)$. We conclude that the total time spent by the routine is $O\left(sn \sum_{j=1}^{r} n_j\right) \leq O(sn^2)$. It follows that the local testability problem of deterministic automata can be solved in $O(sn^2)$ time.

## V. Problems for Future Research

We have presented an $O(sn^2)$ time algorithm for the local testability problem. We have the following questions concerning locally testable automata:

a) Is there a polynomial time algorithm which, given a locally testable deterministic automaton, finds $k$ such that the automaton is properly $k$-testable (i.e., $k$-testable but not $(k-1)$-testable)?

b) As a function of $n$ and $s$, what is the maximum $k$ for which a reduced deterministic finite automaton, with $n$ states and alphabet size $s$, can be properly $k$-testable?

Well after the revised submission of this paper, we discovered that the problem in a) is NP-hard. For question b), it is known that the $k$ of a properly $k$-testable deterministic automaton is bounded above by the size of the semigroup [2], which is exponential in the number of states in the worst case. We conjecture that $k$ is no greater than $O(n^{1.5})$ when the alphabet size is 2, and $O(n^2)$ in the general case. Fig. 7 shows an idea for constructing a properly $k$-testable automaton with large $k$ which approaches this bound. The automaton has 28 states and is properly 127-testable. We think that this is the largest $k$ achievable with 28 states and two input symbols.

In [7], Liu shows interesting properties of $k$th order finite automata, a subclass of finite state automata which can be implemented in terms of feedback-shift-registers. Without giving this definition here, we remark that it is not difficult to show that every locally testable automaton is a $k$th order finite state automaton, for some $k$. This indicates a potential application of our algorithm.

It will be interesting to investigate a generalization of the locally testable languages to their multidimensional versions. Recently, there has been considerable interest in array languages for their application in pattern recognition and image processing [3], in addition to the theoretical interest of its own. It will be much easier to recognize a locally testable image since it does not require global attributes of the entire image. The Turing machines and the linear bounded automata can also be generalized to their two-dimensional tape versions which accept exactly the languages of the phrase structured array grammars and context-sensitive array grammars, respectively [14]. However, for the classes of context-free and regular array languages, there is no such natural extension for both grammars and their acceptors. So our interesting question is: Is there any natural class of machines and/or grammars that characterizes a two-dimensional version of the class of locally testable languages?

Consider the following straightforward extension of the definition of local testability to its two-dimensional version. Let $\Sigma$ be a finite alphabet, and let $\Sigma^{**}$ denote the set of all rectangular arrays over $\Sigma$, including the null array. For $x \in \Sigma^{**}$ and $k \geq 0$, define the following tuples: $C_k(x) = (tl_k(x), tr_k(x), br_k(x), bl_k(x))$, where $tl_k(x)$, $tr_k(x)$, $br_k(x)$, and $bl_k(x)$ are the four $k \times k$ subarrays of $x$, respectively, at the top left, the top right, the bottom right, and the bottom left corners. Define $S_{k+1}(x) = (st_{k+1}(x), sr_{k+1}(x), sb_{k+1}(x), sl_{k+1}(x))$, where $st_{k+1}(x), sr_{k+1}(x), sb_{k+1}(x)$, and $sl_{k+1}(x)$ are, respectively, the set of topmost $(k+1) \times k$ subarrays of $x$, the set of rightmost $k \times (k+1)$ subarrays of $x$, the set of bottommost $(k+1) \times k$ subarrays of $x$, and the set of leftmost $k \times (k+1)$ subarrays of $x$. Finally, let $I_{k+1}(x)$ be the set of all $(k+1) \times (k+1)$ subarrays of $x$, and let $SM_{k+1}(x)$ denote the triple $(C_k(x), S_{k+1}(x), I_{k+1}(x))$. We define locally testable array languages as follows. For $L \subseteq \Sigma^{**}$,

1) $L$ is 0-testable iff it is $\Sigma^{**}$ or $\emptyset$.
2) For a nonnegative integer $k$, $L$ is $(k+1)$-testable, iff, for all $x, y \in \Sigma^{**}$, $SM_{k+1}(x) = SM_{k+1}(y)$ implies that either both $x$ and $y$ are in $L$ or neither is in $L$.
3) $L$ is locally testable iff it is $k$-testable for some $k \geq 0$.

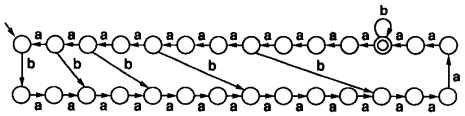


Fig. 7. An automaton with 28 states which is properly 127-testable.

As far as we know, there have been no array grammars or acceptors in the literature that characterize locally testable array languages as defined above. As candidates for processors of locally testable array languages, the finite state array acceptors and the cellular acceptors [14] merit consideration. For the languages of these acceptors we may raise questions similar to those raised for deterministic finite automata, including the local testability problem (i.e., is the language of a given acceptor locally testable?) as well as the questions analogous to 1) and 2) at the beginning of this section. Unfortunately, for array languages, there seems to be no good analog of "finite-state," as was pointed out in [15]. We fear that the local testability problem for array acceptors may be unsolvable.

## Acknowledgment

The authors thank M. Krishnamoorthy for many fruitful discussions and pointers.

## References

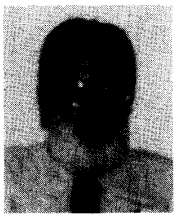
[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley, 1974.
[2] J. Brzozowski and I. Simon, Characterizations of locally testable events," *Discrete Math.,* vol. 4, pp. 243–271, 1973.
[3] H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, Eds., "Graph-grammars and their application to computer science," Lecture Notes in Computer Science 291, Springer-Verlag, 1986.
[4] A. Ginzburg, "About some properties of definite, reverse-definite and related automata," *IEEE Trans. Electron. Comput.,* vol. EC-15, pp. 806–810, 1966.
[5] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979.
[6] H. Hunt and D. Rosenkrantz, "Computational parallels between the regular and context-free languages," *SIAM J. Comput.,* vol. 7, pp. 99–114, 1978.
[7] C. Liu, "$k$th order finite automaton," *IEEE Trans. Electron. Comput.,* vol. EC-13, pp. 738–740, 1964.
[8] R. Martin, *Studies in Feedback-Shift-Register Synthesis of Sequential Machines.* Cambridge, MA: M.I.T. Press, 1969.
[9] R. McNaughton, "Algebraic decision procedures for local testability," *Math. Syst. Theory,* vol. 8, pp. 60–76, 1974.
[10] R. McNaughton and S. Papert, *Counter-free Automata,* Cambridge, MA: M.I.T. Press, 1971.
[11] P. Menon and A. Friedman, "Fault detection in iterative logic arrays," *IEEE Trans. Comput.,* vol. C-20, pp. 524–535, 1971.

[12] M. Minksy and S. Papert, *Perceptrons.* Cambridge, MA: M.I.T. Press, 1969.
[13] M. Perles, M. O. Rabin, and E. Shamir, "The theory of definite automata," *IEEE Trans. Electron. Comput.,* vol. EC-12, pp. 233–243, 1963.
[14] A. Rosenfeld, *Picture Languages.* New York: Academic, 1979.
[15] ——, "Array grammars," Lecture Notes in Computer Science 291, Springer-Verlag, 1986, pp. 67–70.
[16] J. Stern, "Characterizations of some classes of regular events," *Theoret. Comput. Sci.,* vol. 35, pp. 17–42, 1985.
[17] ——, "Complexity of some problems from the theory of automata," *Inform. Contr.,* vol. 66, pp. 163–176, 1985.
[18] Y. Zalcstein, "Locally testable languages," *J. Comput. Syst. Sci.,* vol. 6, pp. 151–167, 1972.

**Sam M. Kim** (M'84) received the B.S. and the M.S. degrees in physics from Kyungbook National University, Korea, and the M.S. and Ph.D. degrees in computer science from University of Minnesota, Minneapolis, in 1982 and 1983, respectively.

From 1969 to 1974 he was an engineer of Control Data, and from 1974 to 1978 a faculty member of Kyungbook National University, Korea. Currently, he is an Associate Professor of Computer Science at Rensselaer Polytechnic Institute. His research interests include systolic system analysis and synthesis, parallel algorithms, theory of computation, and fault-tolerant computing.

Dr. Kim is a member of the IEEE Computer Society, the Association for Computing Machinery, and the European Association for Theoretical Computer Science.



**Robert McNaughton** is Professor Emertius in the Computer Science Department of Rensselaer Polytechnic Institute, where he has been since 1966. After his 1951 Harvard Ph.D. in philosophy, with a specialty in logic, he taught philosophy at the Ohio State University, the University of Michigan, and Stanford University, with research in philosophy, mostly in logic. After 1957 he taught Computer Science at the Moore School at the University of Pennsylvania, M.I.T., and R.P.I. His computer science research has been mostly in automata theory, formal languages, and the combinatorics on words.

Dr. McNaughton is a member of the Association for Computing Machinery, the American Mathematical Society, the Mathematical Association of America, and the Association for Symbolic Logic.

**Robert McCloskey** is on the Computer Science faculty at the University of Scranton. He received the B.S. and M.S. degrees in computer science from the University of Scranton and Rensselaer Polytechnic Institute in 1984 and 1987, respectively.

His research interests are in automata and formal languages, parsing and theory of computation.

Mr. McCloskey is a member of the Association for Computing Machinery.