

On the Minimality of Finite Automata and Stream X-machines for Finite Languages

FLORENTIN IPATE

*Department of Computer Science and Mathematics, University of Pitesti, Str Targu din Vale 1,
0300 Pitesti, Romania
Email: fipate@ifsoft.ro*

A cover automaton of a finite language L is a finite automaton that accepts all words in L and possibly other words that are longer than any word in L . An algorithm for constructing a minimal cover automaton of a finite language L is given in a recent paper. This paper goes a step further by proposing a procedure for constructing all minimal cover automata of a given finite language L . The concept of cover automaton is then generalized to a form of extended finite automaton, the stream X-machine, and the procedure is extended to this more general model.

Received 8 January 2004; revised 30 September 2004

1. INTRODUCTION

Finite automata [1, 2, 3] are widely used in many areas of computing, ranging from lexical analysis to circuit and protocol testing. Finite automata are known to compute regular languages [4, 5]. However, in many applications of finite automata only finite languages are used. The number of states of a finite automaton (FA) that accepts a finite language is at least one more than the length of the longest word in the language and may be exponentially large in this length [6]. On the other hand, if we do not restrict the automaton to accept only the given finite language but allow it to accept extra words that are longer than the longest word in the language, then the number of its states may be significantly reduced. In most applications the maximum length of the words in the language is known and the system can keep track of the length of the words processed, so such an automaton will usually be adequate. This is the idea behind cover automata for finite languages.

Informally, a cover automaton of a finite language L is an FA that accepts all words in L and possibly other words that are longer than any word in L . A minimal cover automaton of L is a cover automaton of L having the least number of states. In many cases, a minimal cover automaton of L has a much smaller size than the minimal automaton that accepts L .

The concept of minimal cover automaton of a finite language is introduced in [6] and it is shown that there may be several minimal cover automata of the same language that are not isomorphic. Furthermore, [6] provides an algorithm that, for a finite language L (given as an FA that accepts L or as a cover automaton of L), constructs a minimal cover automaton of the language. An improved algorithm (in terms of complexity) is also presented in [7].

This paper goes a step further by giving a procedure for constructing all minimal cover automata of a given finite language L . The procedure is then generalized to a form of extended finite automata, called stream X-machines (SXMs).

An SXM is a type of X-machine [8, 9, 10] that describes a system as a finite set of states, each with an internal store called memory, and a number of transitions between the states. A transition is triggered by an input value, produces an output value and may alter the memory. An SXM may be modelled by an FA (the associated FA) in which the arcs are labelled by function names (the processing functions). Thus, SXMs can combine the dynamic features of finite state machines with data structures, thus sharing the benefits of both these worlds. Various case studies [10, 11, 12] have demonstrated the value of the SXM as a specification method, especially for interactive systems. A tool to support the creation of SXM specifications has been constructed [13]. The refinement of SXMs has been investigated and techniques for refining given specifications into more complex, more detailed implementation-oriented versions have been developed [14, 15]. Furthermore, several models of communicating SXMs have been devised and used in real applications [16, 17, 18].

One of the strengths of using SXMs to specify a system is that it is possible to derive test sets from an SXM specification which, if satisfied, guarantee, under certain constraints, the correctness of the implementation with respect to the specification [10, 19, 20, 21]. Among these constraints are the so-called 'design for test conditions' that the SXM specification has to meet: input-completeness and output-distinguishability [10, 19]. The class of SXMs that meet these conditions is therefore of particular interest and has

received special attention. The minimality issue for this class of SXM has been defined and addressed in a recent paper [22].

This paper makes one more step in this direction by defining the l -cover SXM of a given SXM Z as a generalization of the cover automaton of a finite language L . Informally, an l -cover SXM of Z is an SXM Z' that produces an identical response to Z for any sequence of inputs of length at most l .

The paper establishes that, in the context of SXM that meet the 'design for test conditions', the problem of constructing all l -cover SXM of Z can be reduced to constructing the set of all cover automata of the language consisting of all sequences of length at most l accepted by the associated FA of Z .

Since the main motivation for this paper is from the domain of software testing, the paper only addresses finite automata and SXM in which all states are final. When finite automata and SXM specifications are used as the basis for test set generation, all states are naturally considered as being final so that the outputs produced by the application of each input in a sequence can be observed step by step. When testing is performed it is a common practice to display all of these intermediate results, which are normally hidden from the end user of the system. The intermediate outputs are filtered out once the system has passed testing.

The paper is structured as follows. Section 2 introduces the basic concepts of finite automata. Section 3 defines a cover automaton of a finite language and provides a brief survey of some results related to this concept. The procedure for constructing all minimal cover automata of a given finite language is given in Section 4. The theoretical basis for this procedure is also given. Section 5 introduces the concept of SXM. Section 6 defines an l -cover SXM of a given SXM Z and proves that if Z meets the 'design for test conditions' then the procedure given in Section 4 can be used to construct all l -cover SXM of Z . The main contribution of the paper is in Sections 4 and 6.

2. FINITE AUTOMATA

This section introduces finite automata and related concepts and the results to be used later in the paper.

First, the notations used are introduced. For an alphabet A we use A^* to denote the set of finite sequences (words) constructed using elements of A ; ϵ denotes the empty sequence. For $a, b \in A^*$, ab denotes the concatenation of sequences a and b . a^n is defined by $a^0 = \epsilon$ and $a^n = a^{n-1}a$ for $n \geq 1$. For $U, V \subseteq A^*$, $UV = \{ab \mid a \in U, b \in V\}$; U^n is defined by $U^0 = \{\epsilon\}$ and $U^n = U^{n-1}U$ for $n \geq 1$. Also, $U_{\leq n} = \cup_{0 \leq k \leq n} U^k$.

For a sequence $a \in A^*$ $\text{length}(a)$ denotes the number of elements of a , in particular $\text{length}(\epsilon) = 0$. For a finite language $L \subseteq A^*$ $\text{length}(L)$ denotes the length of the longest word(s) in L i.e. $\text{length}(L) = \max\{\text{length}(s) \mid s \in L\}$.

For a (partial) function $f : A \rightarrow B$, $\text{dom}(f)$ denotes the domain of f , i.e. the subset of A for which f is defined.

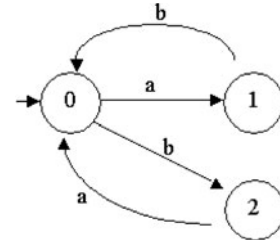


FIGURE 1. A DFA A of L .

DEFINITION 2.1. An FA A is a tuple (Σ, Q, F, I, T) , as follows:

- Σ is the finite input alphabet
- Q is the finite set of states
- F is the (partial) next state function, $F : Q \times \Sigma \rightarrow 2^Q$
- I and T are the sets of initial and terminal states respectively, $I \subseteq Q, T \subseteq Q$.

An FA is usually described by a transition diagram, see Figure 1.

DEFINITION 2.2. A deterministic FA (DFA) is an FA for which the following are true:

- There is one initial state, i.e.

$$I = \{q_0\}$$

- F maps each (state, input) pair into at most one single state, i.e.

$$F : Q \times \Sigma \rightarrow Q.$$

In what follows we will only refer to DFAs in which all states are terminal ($T = Q$), denoted by a tuple (Σ, Q, F, q_0) .

DEFINITION 2.3. The next state function F can be extended to a (partial) function $F^* : Q \times \Sigma^* \rightarrow Q$ defined by:

- $F^*(q, \epsilon) = q, q \in Q$
- $F^*(q, s\sigma) = F(F^*(q, s), \sigma), q \in Q, s \in \Sigma^*, \sigma \in \Sigma$.

DEFINITION 2.4. For $q \in Q$, the language accepted by A in q , denoted by $\mathcal{L}_A(q)$, is defined by:

$$\mathcal{L}_A(q) = \{s \in \Sigma^* \mid (q, s) \in \text{dom}(F^*)\}.$$

The language accepted by A in q_0 is simply called the language accepted by A and is denoted by \mathcal{L}_A .

A language accepted by a DFA is called a regular language. Since in this paper only DFAs in which all states are terminal are considered, the languages computed will have the additional property that if a word is in the language then all the prefixes of the word will also be in the language. Such languages will be called downward-closed.

DEFINITION 2.5. A state $q \in Q$ is called accessible if $\exists s \in \Sigma^*$ such that $F^*(q_0, s) = q$. A is called accessible if $\forall q \in Q, q$ is accessible.

DEFINITION 2.6. Two states $q_1, q_2 \in Q$ are called equivalent if $\mathcal{L}_A(q_1) = \mathcal{L}_A(q_2)$. Otherwise q_1 and q_2 are called distinguishable. A is called reduced if $\forall q_1, q_2 \in Q$ with $q_1 \neq q_2$, q_1 and q_2 are distinguishable.

DEFINITION 2.7. A DFA, A , is called minimal if any other DFA that accepts the same language as A has at least the same number of states as A .

THEOREM 2.1. A DFA, A , is minimal if and only if A is accessible and reduced.

This is a well known result; for a proof see [8].

DEFINITION 2.8. Let $A = (\Sigma, Q, F, q_0)$ and $A' = (\Sigma, Q', F', q'_0)$ be two DFAs over the same input alphabet. Then a function $g : Q \rightarrow Q'$ is called an isomorphism if

- g is bijective
- $g(q_0) = q'_0$
- $g(F(q, \sigma)) = F'(g(q), \sigma)$, $q \in Q, \sigma \in \Sigma$.

THEOREM 2.2. For two minimal DFAs, A and A' , $\mathcal{L}_A = \mathcal{L}_{A'}$ if and only if A and A' are isomorphic.

This is a well known result, for a proof see [8]. Techniques for constructing the minimal DFA that accepts a certain language also exist, for more details see [8] or [3].

3. DETERMINISTIC FINITE COVER AUTOMATA—PRELIMINARIES

This section introduces the concept of (minimal) deterministic finite cover automaton (DFCA) of a finite language. The definitions and results are largely from [6].

A DFCA of a finite language L is a DFA that accepts all words in L and possibly other words that are longer than any word in L . As already stated, only DFAs in which all states are terminal and downward-closed finite languages are considered in this paper.

DEFINITION 3.1. Let $A = (\Sigma, Q, F, q_0)$ be a DFA, $L \subseteq \Sigma^*$ a finite language and $l = \text{length}(L)$. Then A is called a DFCA of L if $\mathcal{L}_A \cap \Sigma_{\leq l} = L$.

EXAMPLE 3.1. If $\Sigma = \{a, b\}$, $L = \{\epsilon, a, b, ab, ba\}$ and A is as represented in Figure 1 (where the initial state 0 is pointed at by an incoming arrow) then $\mathcal{L}_A \cap \Sigma_{\leq 2} = L$, so A is a DFCA of L .

DEFINITION 3.2. A DFCA, A , of a finite language L is called minimal if for any DFCA, A' , of L , the number of states of A is less than or equal to the number of states of A' .

Two similarity relations defined in [6] are used to characterize and construct a minimal DFCA for a finite language L :

- a similarity relation between sequences of inputs w.r.t. the language L
- an l -similarity relation between the states of a DFA, where l is the length of the longest sequence(s) in L .

Their formal definitions are now given.

DEFINITION 3.3. Let Σ be an alphabet, $L \subseteq \Sigma^*$ a finite language and $l = \text{length}(L)$ the length of the longest word(s) in L . Then \sim_L , a relation on Σ^* , is defined by: $s \sim_L t$ if $\forall x \in \Sigma^*$ with $\text{length}(sx) \leq l$ and $\text{length}(tx) \leq l$, $sx \in L$ if and only if $tx \in L$.

We say that s is similar to t w.r.t. L . The relation \sim_L is called the similarity relation on Σ^* w.r.t. L . When $s \sim_L t$ does not hold we write $s \not\sim_L t$.

Note 3.1. The similarity relation w.r.t. L is reflexive and symmetric but not transitive. For example, if $L = \{\epsilon, a, b, ab, ba\}$, then $a \sim_L ab, b \sim_L ab$, but $a \not\sim_L b$.

However, we can prove the following result that will be used later in the paper.

LEMMA 3.1. Let Σ be an alphabet, $L \subseteq \Sigma^*$ a finite language and $s, t, u \in \Sigma^*$. If $s \sim_L t, t \sim_L u$ and $\text{length}(t) \leq \max\{\text{length}(s), \text{length}(u)\}$ then $s \sim_L u$.

Proof. From $s \sim_L t$ it follows that $\forall x \in \Sigma^*$ with $\text{length}(x) \leq l - \max\{\text{length}(s), \text{length}(t)\}$, $sx \in L$ if and only if $tx \in L$. Similarly, from $t \sim_L u$ it follows that $\forall x \in \Sigma^*$ with $\text{length}(x) \leq l - \max\{\text{length}(t), \text{length}(u)\}$, $tx \in L$ if and only if $ux \in L$. Hence $\forall x \in \Sigma^*$ with $\text{length}(x) \leq l - \max\{\text{length}(s), \text{length}(t), \text{length}(u)\}$, $sx \in L$ if and only if $ux \in L$. Since $\text{length}(t) \leq \max\{\text{length}(s), \text{length}(u)\}$, we have that $\max\{\text{length}(s), \text{length}(t), \text{length}(u)\} = \max\{\text{length}(s), \text{length}(u)\}$, so $s \sim_L u$. \square

DEFINITION 3.4. Let $A = (\Sigma, Q, F, q_0)$ be an accessible DFA. For each state $q \in Q$ we define $\text{level}_A(q)$ as the length of the shortest path(s) from q_0 to q , i.e. $\text{level}_A(q) = \min\{\text{length}(s) \mid s \in \Sigma^*, F^*(q_0, s) = q\}$.

For the DFA in Example 3.1 we have $\text{level}_A(0) = 0, \text{level}_A(1) = \text{level}_A(2) = 1$.

DEFINITION 3.5. Let $A = (\Sigma, Q, F, q_0)$ be an accessible DFA. For each state $q \in Q$ we define $x_A(q)$ as the minimum path from q_0 to q , i.e. $x_A(q) = \min\{s \mid s \in \Sigma^*, F^*(q_0, s) = q\}$, where the minimum is taken according to the quasi-lexicographical order on Σ^* .

Note 3.2. If $\Sigma = \{\sigma_1, \dots, \sigma_n\}$ is an ordered set, $n > 0$, then the quasi-lexicographical order on Σ^* , denoted by $<$, is defined by: $x < y$ if $\text{length}(x) < \text{length}(y)$ or $\text{length}(x) = \text{length}(y)$ and $x = z\sigma_i v, y = z\sigma_j u, i < j$, for some $z, u, v \in \Sigma^*$ and $1 \leq i, j \leq n$.

For the DFA in Example 3.1 we have $x_A(0) = \epsilon, x_A(1) = a$ and $x_A(2) = b$.

DEFINITION 3.6. Let $A = (\Sigma, Q, F, q_0)$ be an accessible DFA and let $l \geq 0$. Then \sim_A^l is a relation on Q defined by: $p \sim_A^l q$ if $\forall x \in \Sigma^*$ with $\text{length}(x) \leq l - \max\{\text{level}_A(p), \text{level}_A(q)\}$, $x \in \mathcal{L}_A(p)$ if and only if $x \in \mathcal{L}_A(q)$.

We say that p is l -similar to q w.r.t. A . The relation \sim_A^l is called the l -similarity relation on Q w.r.t. A . When $p \not\sim_A^l q$ does not hold we write $p \not\sim_A^l q$.

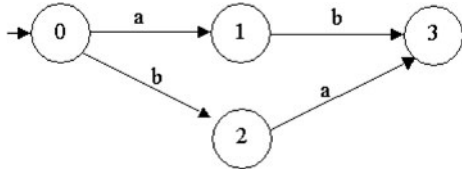


FIGURE 2. The minimal DFA B that accepts L .

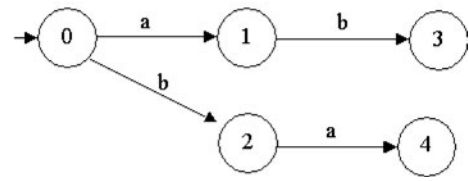


FIGURE 3. The canonical DFA A_L of L .

Note 3.3. As for the similarity relation on Σ^* w.r.t. L , the l -similarity relation on Q w.r.t. A is reflexive and symmetric but not transitive. For B as represented in Figure 2 and $l = 2$ we have $1 \sim_B^l 3$, $2 \sim_B^l 3$, but $1 \not\sim_B^l 2$.

The following lemma establishes the link between the two similarity relations.

LEMMA 3.2 [6]. *Let $L \subseteq \Sigma^*$ be a finite language, $l = \text{length}(L)$ and A an accessible DFCA of L . The following are equivalent:*

- (i) $p \sim_A^l q$
- (ii) $x_A(p) \sim_L x_A(q)$.

DEFINITION 3.7. *An accessible DFA, $A = (\Sigma, Q, F, q_0)$, is called l -reduced if $\forall p, q \in Q$ with $p \neq q$, $p \not\sim_A^l q$.*

That is, an accessible DFA is l -reduced if any two distinct states are not l -similar w.r.t. A .

The following theorem identifies the necessary conditions for a DFCA to be minimal.

THEOREM 3.1 [6]. *If $A = (\Sigma, Q, F, q_0)$ is a minimal DFCA of a finite language L and $l = \text{length}(L)$ then A is accessible and l -reduced.*

COROLLARY 3.1 [6]. *A minimal DFCA of a finite language L is also a minimal DFA.*

The converse is not however true as is illustrated by the following example.

EXAMPLE 3.2. If $\Sigma = \{a, b\}$ and $L = \{\epsilon, a, b, ab, ba\}$ then B as represented in Figure 2 is the minimal DFA that accepts L . However, B is not a minimal DFCA of L since A in Figure 1 is a DFCA of L that has fewer states than B .

An algorithm that determines the l -similarity relation between the states of an accessible DFA is given in [6]. On the basis of this algorithm a procedure for constructing a minimal DFCA of a finite language L (given as a DFA that accepts L or as a DFCA of L) is then developed.

4. CONSTRUCTION OF ALL MINIMAL DFCA'S OF L

In this section we provide a more general procedure for constructing all minimal DFCA's of L , assuming that the l -similarity relation between the states of an accessible DFA is known.

We first set out the theoretical basis for the construction of these.

DEFINITION 4.1. *Let $L \subseteq \Sigma^*$ be a finite language. Then $A_L = (\Sigma, L, F_L, \epsilon)$, where F_L is defined by $F_L(s, \sigma) = s\sigma$, $s \in L, \sigma \in \Sigma$ such that $s\sigma \in L$, is called the canonical DFA of L .*

That is, the canonical DFA of L has a state for every input sequence in L .

Note 4.1. Clearly, $\mathcal{L}_{A_L} = L$ and the similarity relations \sim_L and $\sim_{A_L}^l$, where $l = \text{length}(L)$, coincide. For simplicity, in what follows we will use \sim_L to denote both the similarity of input sequences w.r.t. L and the l -similarity of states w.r.t. A_L .

EXAMPLE 4.1. If $\Sigma = \{a, b\}$ and $L = \{\epsilon, a, b, ab, ba\}$ then A_L as represented in Figure 3 is the canonical DFA of L . For clarity, in Figure 3 the names of states are numbers rather than sequences of inputs.

DEFINITION 4.2. *Let $L \subseteq \Sigma^*$ be a finite language, $A_L = (\Sigma, L, F_L, \epsilon)$ the canonical DFA of L and $A = (\Sigma, Q, F, q_0)$ a DFCA of L . Then we define the function $\pi_A : L \rightarrow Q$ by $\pi_A(s) = F^*(q_0, s)$, $s \in L$. We call π_A the projection of A_L on A .*

LEMMA 4.1. *For all $s, t \in L$, if $\pi_A(s) = \pi_A(t)$ then $s \sim_L t$.*

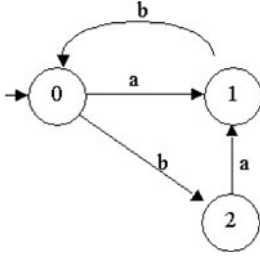
Proof. If $\pi_A(s) = \pi_A(t)$ then $F^*(q_0, s) = F^*(q_0, t)$, so $\forall x \in \Sigma^*$, $sx \in \mathcal{L}_A$ iff $tx \in \mathcal{L}_A$. Since A is a DFCA of L , we have $\mathcal{L}_A \cap \Sigma_{\leq l} = L$, so $\forall x \in \Sigma^*$ with $\text{length}(sx) \leq l$ and $\text{length}(tx) \leq l$, $sx \in L$ iff $tx \in L$. \square

LEMMA 4.2. *If A is a minimal DFCA of L then π_A is a surjective function.*

Proof. Assume otherwise and let $q \in Q$ such that $x_A(q) \notin L$. Since A is a DFCA of L it follows that $\text{length}(x_A(q)) > \text{length}(L)$. Then if q is removed from A , the remaining automaton will still be a DFCA of L . Thus A is not a minimal DFCA of L , which contradicts the original assumption. \square

DEFINITION 4.3. *Let $A = (\Sigma, Q, F, q_0)$ be an accessible DFA and h an equivalence relation on Q . Then for each $q \in Q$ we define $y_A^h(q)$ as the minimum path from q_0 that reaches a state in the same equivalence class as q , i.e. $y_A^h(q) = \min\{x_A(p) \mid q h p\}$, where the minimum is taken according to the quasi-lexicographical order.*

EXAMPLE 4.2. For A_L as represented in Figure 3 and $h = \{\{0, 3\}, \{1, 4\}, \{2\}\}$ we have $y_{A_L}^h(0) = y_{A_L}^h(3) = \min\{\epsilon, ab\} = \epsilon$, $y_{A_L}^h(1) = y_{A_L}^h(4) = \min\{a, ba\} = a$ and $y_{A_L}^h(2) = b$.


 FIGURE 4. A_L^h .

DEFINITION 4.4. Let $A = (\Sigma, Q, F, q_0)$ be an accessible DFA and h an equivalence relation on Q . Then we define a DFA $A^h = (\Sigma, Q^h, F^h, q_0)$ by:

- $Q^h = \{F^*(q_0, y_A^h(q)) \mid q \in Q\}$
- $F^h : Q^h \times \Sigma \rightarrow Q^h$ is a (partial) function defined by $F^h(q, \sigma) = F^*(q_0, y_A^h(F(q, \sigma)))$, $q \in Q^h, \sigma \in \Sigma$.

That is, for each equivalence class of h we identify the states whose access paths from q_0 are minimum; Q^h is the set of all these states. Then A^h is constructed by replacing each transition $F(q, \sigma) = p$ in A , $q \in Q^h, p \in Q$ with a transition $F(q, \sigma) = p'$, where $p' \in Q^h$ is such that $p h p'$.

EXAMPLE 4.3. For A_L as represented in Figure 3 and $h = \{\{0, 3\}, \{1, 4\}, \{2\}\}$, we have $F_L^*(0, y_{A_L}^h(3)) = F_L^*(0, y_{A_L}^h(0)) = 0$ and $F_L^*(0, y_{A_L}^h(4)) = F_L^*(0, y_{A_L}^h(1)) = 1$, so A_L^h is as represented in Figure 4.

The following lemma shows that any minimal DFCA of a finite language L can be constructed as A_L^h for some equivalence relation h on the states of A_L .

LEMMA 4.3. Let $L \subseteq \Sigma^*$ be a finite language, A_L the canonical DFA of L , $A = (\Sigma, Q, F, q_0)$ a minimal DFCA of L , π_A the projection of A_L on A and h the equivalence relation induced by π_A on L . Then A and A_L^h are isomorphic.

Proof. By Definition 4.3 and Definition 4.4, we have $y_{A_L}^h(s) = \min\{x \in L \mid F^*(q_0, s) = F^*(q_0, x)\}$ and $A_L^h = (\Sigma, L^h, F_L^h, \epsilon)$, where

- $L^h = \{y_{A_L}^h(s) \mid s \in L\}$
- $F_L^h : L^h \times \Sigma \rightarrow L^h$ is defined by $F_L^h(s, \sigma) = y_{A_L}^h(s\sigma)$, $s \in L^h, \sigma \in \Sigma$ such that $s\sigma \in L$.

Let $f : L^h \rightarrow Q$ be the restriction of π_A to L^h , i.e. f is defined by $f(s) = F^*(q_0, s)$, $s \in L^h$. Since A is a minimal DFCA of L , π_A is surjective, so f is a bijective function.

Let $s, s' \in L^h$ and $\sigma \in \Sigma$. Then $y_{A_L}^h(s\sigma) = s'$ iff $F^*(q_0, s\sigma) = F^*(q_0, s')$. Therefore $F_L^h(s, \sigma) = s'$ iff $F(F^*(q_0, s), \sigma) = F^*(q_0, s')$, so $F_L^h(s, \sigma) = s'$ iff $F(f(s), \sigma) = f(s')$. Hence f is an isomorphism. \square

Conversely, A_L^h is a minimal DFCA of L if h belongs to a special class of equivalence relations on L , as shown below.

DEFINITION 4.5. Let $L \subseteq \Sigma^*$ be a finite language. Then an equivalence relation on L is called a similarity equivalence

relation on L (SER on L) if whenever $s h t$ we have $s \sim_L t$ for $s, t \in L$.

EXAMPLE 4.4. $h = \{\{\epsilon, ab\}, \{a, ba\}, \{b\}\}$ is an SER on $L = \{\epsilon, a, b, ab, ba\}$. If L is given as the canonical DFA represented in Figure 3, where the names of states are given as numbers rather than sequences of inputs, then h will be written as $h = \{\{0, 3\}, \{1, 4\}, \{2\}\}$.

LEMMA 4.4. If h is a SER on L then A_L^h is a DFCA of L .

Proof. Let $A_L^h = (\Sigma, L^h, F_L^h, \epsilon)$ and let L_0 be the language computed by A_L^h . We have to prove that $L_0 \cap \Sigma_{\leq l} = L$, where $l = \text{length}(L)$.

From Definition 4.4 it follows easily that $L \subseteq L_0 \cap \Sigma_{\leq l}$ so it remains to show that $L_0 \cap \Sigma_{\leq l} \subseteq L$. If we assume otherwise then there exist $s \in \Sigma^*$ with $\text{length}(s) < l$ and $\sigma \in \Sigma$ such that $s \in L$, $s\sigma \notin L$ and $s\sigma \in L_0$. Since $s \in L$ there exists $t \in L^h$ with $\text{length}(t) \leq \text{length}(s)$ such that $F_L^{h*}(\epsilon, s) = t$ and $s h t$. Since $s\sigma \in L_0$, $F_L^h(t, \sigma)$ is defined, $F_L(t, \sigma)$ is defined, so $t\sigma \in L$. As $s h t$ and h is an SER on L we have $s \sim_L t$ with $\text{length}(t) \leq \text{length}(s) < l$, so from $t\sigma \in L$ it follows that $s\sigma \in L$, which contradicts our assumption. \square

DEFINITION 4.6.

- An SER h on L is called proper if whenever $s h t$ does not hold, $s, t \in L$, $\exists x, y \in L$ such that $s h x, t h y$ and $x \approx_L y$.
- An SER h on L is called maximal if for any SER h' on L , the number of equivalence classes of h is less than or equal to the number of equivalence classes of h' .

That is, an SER on L is proper if any two distinct equivalence classes contain at least one pair of sequences that are not similar w.r.t. L . A maximal SER on L has the least possible number of equivalence classes.

EXAMPLE 4.5. For A_L as represented in Figure 3, $h = \{\{0, 3\}, \{1, 4\}, \{2\}\}$ is a proper and maximal SER on L .

DEFINITION 4.7. Let $L \subseteq \Sigma^*$ be a finite language. Then a set $X = \{x_1, \dots, x_n\} \subseteq L$ is called a dissimilar set of L if $x_i \approx_L x_j$ for $1 \leq i, j \leq n, i \neq j$.

LEMMA 4.5. Let $L \subseteq \Sigma^*$ be a finite language and h an SER on L . Then the following are equivalent:

- h is a maximal SER on L
- h is a proper SER on L
- if c_1, \dots, c_n are the equivalence classes of h and $x_i = \min c_i$, $1 \leq i \leq n$, where the minimum is taken according to the quasi-lexicographical order, then $X = \{x_1, \dots, x_n\}$ is a dissimilar set of L .

Furthermore, the set X is the same for any proper (maximal) SER on L .

Proof. (i) \Rightarrow (ii): Assume h is not proper. Then there exist $s, t \in L$ that are not h -equivalent such that $\forall x, y \in L$ with $s h x$ and $t h y$ we have $x \sim_L y$. Then we can construct a new SER h' on L from h by merging the equivalence classes of s and t , so h is not a maximal SER on L , which contradicts our original assumption.

(ii) \Rightarrow (iii): Let c_i and c_j be two equivalence classes of h , $x_i = \min c_i$ and $x_j = \min c_j$. Assume $x_i \sim_L x_j$ and let $s \in c_i$ and $t \in c_j$ be arbitrarily chosen. Then $s \sim_L x_i$ and $t \sim_L x_j$. From Lemma 3.1, since $x_i \sim_L x_j$, $s \sim_L x_i$ and $\text{length}(x_i) \leq \text{length}(s)$ we have that $s \sim_L x_j$. Similarly, from $s \sim_L x_j$, $t \sim_L x_j$ and $\text{length}(x_j) \leq \text{length}(t)$ it follows that $s \sim_L t$, which contradicts the fact that h is a proper SER.

(iii) \Rightarrow (i): Assume h is not a maximal SER and let h' be an SER on L having fewer equivalence classes than h . Then there exist i and j , $1 \leq i, j \leq n$, $i \neq j$, such that x_i and x_j belong to the same equivalence class of h' . Since h' is an SER it follows that $x_i \sim_L x_j$, which contradicts the fact that X is a dissimilar set of L .

Let h' be another proper (maximal) SER on L , c'_1, \dots, c'_n the equivalence classes of h' , $x'_i = \min c'_i$, $1 \leq i \leq n$, and $X' = \{x'_1, \dots, x'_n\}$. Without loss of generality we can assume that $x_1 < x_2 < \dots < x_n$ and $x'_1 < x'_2 < \dots < x'_n$. Obviously $x_1 = x'_1 = \epsilon$. Assume $X \neq X'$. Then there exists k , $2 \leq k \leq n$, such that $x_i = x'_i$, $1 \leq i \leq k-1$, and $x_k \neq x'_k$. Assume (without loss of generality) that $x_k < x'_k$. Then $x_k \in c'_j$ for some j , $1 \leq j \leq k-1$, so $x_k \sim_L x_j$, which contradicts the fact that X is a dissimilar set of L . \square

We denote by H_L the set of all proper (maximal) SERs on L . The set X is called the proper dissimilar set of L .

We can now assemble the result which is our goal.

THEOREM 4.1. *Let $L \subseteq \Sigma^*$ be a finite language and A_L the canonical DFA of L . Then A is a minimal DFCA of L if and only if there exists $h \in H_L$ such that A and A_L^h are isomorphic.*

Proof. ‘only if:’ From Lemma 4.3 it follows that A and A_L^h are isomorphic, where h is the equivalence relation induced by π_A on L . From Lemma 4.1, if $\pi_A(s) = \pi_A(t)$ then $s \sim_L t$, so h is an SER on L . Assume h is not a maximal SER on L . Then there exists an SER h' on L with fewer equivalence classes than h . It is easy to see that in this case $A_L^{h'}$ is a DFCA of L that has fewer states than A , so A is not a minimal DFCA of L , which contradicts the original hypothesis.

‘if:’ From Lemma 4.4 it follows that A is a DFCA of L . Assume A is not a minimal DFCA of L and let A' be a minimal DFCA of L . Then there exists $h' \in H_L$ such that A' and $A_L^{h'}$ are isomorphic, so h' has fewer equivalence classes than h . Therefore $h \notin H_L$, which contradicts our original assumption. \square

An algorithm for determining all minimal DFCA's of a finite language L is given below. It comprises the following steps:

- (i) Construct A_L , the canonical DFA of L .
- (ii) Determine the similarity relation \sim_L between the states of A_L . For this step the algorithm provided in [6] can be used.
- (iii) Construct the proper dissimilar set $X = \{x_1, \dots, x_n\}$ of L by examining the elements of L in quasi-lexicographical order: $x_1 = \epsilon$ and for $2 \leq k \leq n$,

x_k is the first element examined such that $x_i \sim_L x_k$ for $1 \leq i \leq k-1$.

- (iv) Determine H_L , the set of all proper (maximal) SERs on L . A proper (maximal) SER h on L is constructed by placing the elements of $L \setminus X$ into the n equivalence classes determined by x_1, \dots, x_n ; $s \in L \setminus X$ may be placed into the class c_i determined by x_i , $1 \leq i \leq n$, if and only if $s \sim_L x_i$.
- (v) For each $h \in H_L$, construct A_L^h . The set of all minimal DFCA's of L is $\{A_L^h \mid h \in H_L\}$.

EXAMPLE 4.6. Consider L and A_L as in Example 4.1. Then $3 \sim_L 0$, $3 \sim_L 1$, $3 \sim_L 2$, $3 \sim_L 4$, $4 \sim_L 0$, $4 \sim_L 1$, $4 \sim_L 2$ and all the other pairs of states are not similar w.r.t. L . Thus $n = 3$ and $X = \{0, 1, 2\}$. Since 3 and 4 are similar, w.r.t. L , to all the states of A_L , each may be placed in any of the classes determined by 1, 2 and 3. Therefore H_L contains the following equivalence relations:

$$\begin{aligned} h_1 &= \{\{0, 3\}, \{1, 4\}, \{2\}\}, & \text{i.e. } 3 \in c_0 \text{ and } 4 \in c_1, \\ h_2 &= \{\{0, 4\}, \{1, 3\}, \{2\}\}, & \text{i.e. } 4 \in c_0 \text{ and } 3 \in c_1, \\ h_3 &= \{\{0, 3\}, \{1\}, \{2, 4\}\}, & \text{i.e. } 3 \in c_0 \text{ and } 4 \in c_2, \\ h_4 &= \{\{0, 4\}, \{1\}, \{2, 3\}\}, & \text{i.e. } 4 \in c_0 \text{ and } 3 \in c_2, \\ h_5 &= \{\{0\}, \{1, 3\}, \{2, 4\}\}, & \text{i.e. } 3 \in c_1 \text{ and } 4 \in c_2, \\ h_6 &= \{\{0\}, \{1, 4\}, \{2, 3\}\}, & \text{i.e. } 4 \in c_1 \text{ and } 3 \in c_2, \\ h_7 &= \{\{0, 3, 4\}, \{1\}, \{2\}\}, & \text{i.e. } 3, 4 \in c_0, \\ h_8 &= \{\{0\}, \{1, 3, 4\}, \{2\}\}, & \text{i.e. } 3, 4 \in c_1, \\ h_9 &= \{\{0\}, \{1\}, \{2, 3, 4\}\}, & \text{i.e. } 3, 4 \in c_2. \end{aligned}$$

Then Figure 5 represents all minimal DFCA's of L .

In effect, the first phase of the algorithm (first three steps) computes a minimal DFCA of L . If N denotes the number of words in L and l the length of its longest word(s), then the complexity of the first step of the algorithm is $O(N)$, the complexity of the second is $O((N \cdot l)^2)$ [6], while the complexity of the third step is $O(N)$. In general $l \ll N$, so the complexity of the first phase is $O(N^2)$. The second phase (the last two steps) computes the remaining minimal DFCA's of L . The amount of work required in this phase obviously depends on k , the number of all minimal DFCA's of L . The upper bound for k is n^{N-n} . This bound is achieved when each element of $L \setminus X$ may be placed in any of the classes determined by the elements of X , as is the case in Example 4.6. The complexity of the fourth step is n^{N-n} . The amount of work required in the last step is proportional to k , n and p , the number of elements of the input alphabet Σ , so the complexity of this step is $n^{N-n+1} \cdot p$. Thus, the overall complexity of the algorithm is exponential in N , but this is because the maximum number of minimal DFCA's of L is itself exponential in N . In practice, the actual number of minimal DFCA's constructed will be limited by time and space constraints.

Very often, one may wish to choose from among many different, functionally equivalent, specifications the one whose architecture suits best the purpose for which the system is to be constructed. For instance, many finite state machine testing and failure location techniques assume that the specification is strongly connected [23]. An FA is called strongly connected if, given any ordered pair of states

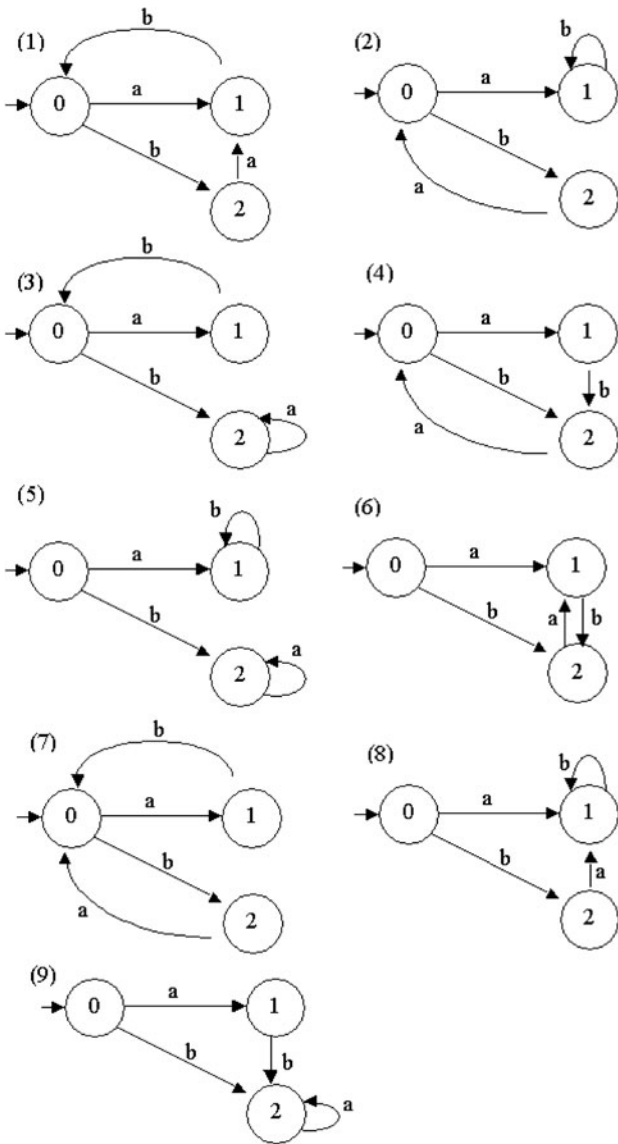


FIGURE 5. $A_L^{h_i}, 1 \leq i \leq 9$.

(q_i, q_j) , there is a sequence of transitions that moves it from q_i to q_j . An accessible automaton is strongly connected if and only if there is a sequence of transitions from any given non-initial state q to the initial state q_0 . Thus, if strongly connected minimal DFCA's are sought, the algorithm can be modified so as to construct first the proper (maximal) SERs for which the class determined by the initial state is the largest—this is because any further element which is placed in the same class as the initial state will give rise to at least one more transition to the initial state in the resulting minimal DFCA. If this strategy is applied to our example h_7 will be constructed first, then h_1, h_2, h_3 and h_4 . It can be easily observed that the minimal DFCA's induced by h_7, h_1 and h_4 are strongly connected.

A more efficient algorithm, of time complexity $O(N \log N)$, that finds a minimal DFCA of L has recently been published [24]. Future work will assess how effectively this new algorithm can replace the first three steps [of complexity

$O(N^2)$] of the above procedure. However, for N sufficiently large and the total number of minimal DFCA's close to the upper bound, the inclusion of this improved algorithm will have little effect on the overall performance of the proposed procedure.

For any given $l \geq 1$, the number of states n of a minimal DFCA can be as low as 1 (i.e. when $L = \Sigma_{\leq l}$) and as high as $N - 1$ (i.e. when $L = \{\epsilon, a, \dots, a^{l-1}, a^{l-1}b\}$, where $a, b \in \Sigma$).

5. STREAM X-MACHINES

The remainder of the paper generalizes the concept of finite cover automaton to a type of extended FA, namely the SXM. The following section introduces this model and other basic concepts related to it.

In essence an X-machine is like a finite state machine but with one important difference. Instead of using abstract symbols, the labels of the transitions are (partial) functions that operate on a basic data set X . The set of these (partial) functions, Φ , is called the type of the machine and represents the elementary operations that the machine is capable of performing.

The computation of the machine starts in a given initial state (control state) and a given state of the system's underlying data type X (the data state). In Figure 6, for example, there are a number of paths that can be traced from the initial state and each edge is labelled by a function: ϕ_1, ϕ_2 etc. Sequences of functions are thus derived from each path in the state space and these may be composed to produce a function that may be defined on the data state. This is then applied to the value x providing that the composed function is defined on x . This gives a new value, $x \in X$ for the data state and a new control state. Usually, the machine is deterministic so that at any instant only one possible function is defined (i.e. the domains of the functions that emerge from any state are mutually disjoint).

Those X-machines in which all data are triples consisting of a stream of input symbols, a stream of output symbols and an internal memory value are called SXMs and are defined formally next. The basic idea is that the machine has some internal memory, M , and the stream of inputs determines, depending on the current state of control and the current state of the memory, the next control state, the next memory state and the output value.

DEFINITION 5.1. An SXM is a tuple

$$Z = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0),$$

where:

- Σ and Γ are finite sets called the input alphabet and output alphabet respectively
- Q is the finite set of states
- M is a (possibly) infinite set called memory
- Φ is the type of Z , a finite set of non-empty basic processing functions that the machine can use, having the form

$$\phi : M \times \Sigma \rightarrow \Gamma \times M$$

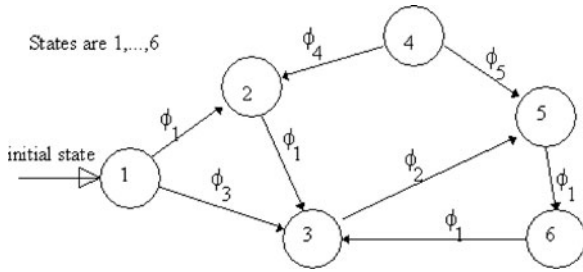


FIGURE 6. The state transition diagram of an X-machine.

- F is the (partial) next state function,

$$F : Q \times \Phi \rightarrow 2^Q$$

As for finite automata, F is usually described by a state-transition diagram.

- I and T are the sets of initial and terminal states respectively,

$$I \subseteq Q, T \subseteq Q$$

- m_0 is the initial memory value,

$$m_0 \in M.$$

Thus, SXMs are X-machines for which the basic processing functions have the form $\phi : M \times \Sigma \rightarrow \Gamma \times M$, i.e. each such function will read an input symbol, discard it and produce an output symbol while (possibly) changing the value of the memory.

It is sometimes helpful to think of an X-machine as an FA with the arcs labelled by functions from the type Φ . The automaton $A_Z = (\Phi, Q, F, I, T)$ over the alphabet Φ is called the associated FA of Z .

DEFINITION 5.2. Given a sequence $p \in \Phi^*$, p induces the (partial) function

$$|p| : M \times \Sigma^* \rightarrow \Gamma^* \times M$$

defined as follows:

- $|\epsilon|(m, \epsilon) = (\epsilon, m), \forall m \in M$
- $\forall p \in \Phi^*, \phi \in \Phi, |p\phi|(m, s\sigma) = (g\gamma, m'), \forall m, m' \in M, s \in \Sigma^*, g \in \Gamma^*, \sigma \in \Sigma, \gamma \in \Gamma$ such that $\exists m'' \in M$ with $|p|(m, s) = (m'', g)$ and $\phi(m'', \sigma) = (\gamma, m')$.

Thus $|p|$ shows the correspondence between a (memory, input string) pair and the (output string, memory) pair produced by the application, in turn, of the processing functions in the sequence p .

A deterministic SXM (DSXM) is one in which there is at most one possible transition for any triplet $q \in Q, m \in M, \sigma \in \Sigma$. This is now defined.

DEFINITION 5.3. An SXM Z is called deterministic if:

- The associated FA of the machine is deterministic, i.e.
— Z has only one initial state, i.e.

$$I = \{q_0\}$$

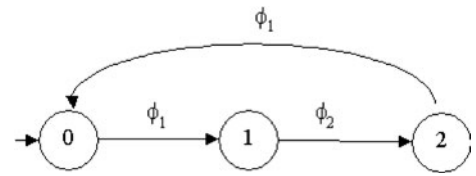


FIGURE 7. The state transition diagram of the DSXM Z .

—The next state function of Z maps each pair state/processing function onto at most one state, i.e.

$$F : Q \times \Phi \rightarrow Q$$

- Any two distinct processing functions that label arcs emerging from the same state have disjoint domains, i.e. $\forall \phi_1, \phi_2 \in \Phi$ if $\exists q \in Q$ with $(q, \phi_1), (q, \phi_2) \in \text{dom}(F)$ then $\phi_1 \neq \phi_2$ or $\text{dom}(\phi_1) \cap \text{dom}(\phi_2) = \emptyset$.

DSXM specifications are often used as a basis for testing [10, 20], so it is normal to consider all states as being terminal, i.e. $T = Q$. Furthermore, this is not normally a restriction when considering interactive systems.

Thus, in what follows, we will refer to DSXMs in which all states are terminal, denoted by a tuple $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$. The associated FA is then a tuple $A_Z = (\Sigma, Q, F, q_0)$.

EXAMPLE 5.1. Consider $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ with $\Sigma = \{a, b\}$, $\Gamma = \{x, y, z, w\}$, $Q = \{0, 1, 2\}$, $M = \{0, 1\}$, $m_0 = 0$, $\Phi = \{\phi_1, \phi_2\}$, F as represented in Figure 7 and $\phi_1, \phi_2 : M \times \Sigma \rightarrow \Gamma \times M$ defined by:

$$\begin{aligned} \phi_1(0, a) &= (x, 0), \\ \phi_1(1, a) &= (y, 1); \\ \phi_2(0, a) &= (z, 1), \\ \phi_2(1, b) &= (w, 0). \end{aligned}$$

Then Z is a DSXM.

A machine computation takes the form of a traversal of all sequences of arcs in the state space from the initial state and the application, in turn, of the arc labels (which represent basic processing functions) to the initial memory value. The correspondence between the input sequence applied to the machine and the output produced gives rise to the function computed by the machine, as defined next.

DEFINITION 5.4. Given a DSXM Z , the (partial) function $f_Z : \Sigma^* \rightarrow \Gamma^*$ defined by $f_Z(s) = g$ if $\exists p \in \Phi^*, m \in M$, such that $p \in \mathcal{L}_{A_Z}$ and $|p|(m_0, s) = (g, m)$, is called the function computed by Z . We say that Z computes f_Z .

Note that f_Z is a function since Z is deterministic. However, in general, a (non-deterministic) SXM may compute a relation rather than a function since the application of an input sequence may produce more than one output sequence. Here, however, only the deterministic case is considered.

DEFINITION 5.5. Φ is called *input-complete* if $\forall \phi \in \Phi, m \in M, \exists \sigma \in \Sigma$ such that $(m, \sigma) \in \text{dom}(\phi)$.

This condition ensures that any processing function can be exercised from any memory value using appropriate input symbols.

DEFINITION 5.6. Φ is called *output-distinguishable* if $\forall \phi_1, \phi_2 \in \Phi, (\exists m, m_1, m_2 \in M, \sigma \in \Sigma, \gamma \in \Gamma$ with $\phi_1(m, \sigma) = (\gamma, m_1)$ and $\phi_2(m, \sigma) = (\gamma, m_2)) \implies (\phi_1 = \phi_2)$.

This says that we must be able to distinguish between any two different processing functions (the ϕ s) by examining outputs. If we cannot then we will not always be able to tell them apart.

It can be easily observed that for Z as in Example 5.1, Φ is both output-distinguishable and input-complete.

These two conditions (output-distinguishability and input-completeness) are generally known as ‘design for test conditions’ [10, 19] and are assumed to be met by the specification if the DSXM-based test set generation technique is to be successfully used [10, 19]. The output-distinguishability condition ensures that any processing function can be identified from the machine computation by examining the outputs produced. The input-completeness condition ensures that all sequences of processing functions in the associated FA can be exercised using appropriate inputs, so they can be tested against the implementation.

6. MINIMAL l -COVER DSXMS

In this section an l -cover DSXM of a given DSXM Z as a DSXM Z' that produces an identical response to Z for any sequence of inputs of length at most l is defined. It is also shown that, if Z satisfies the ‘design for test conditions’, then the construction of all l -cover DSXMs of Z can be reduced to the construction of all DFCA of all cover automata of the language comprising all sequences of length at most l accepted by the associated FA of Z . As already stated, only DSXMs in which all states are terminal are considered in this paper.

DEFINITION 6.1. Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $Z' = (\Sigma, \Gamma, Q', M, \Phi, F', q'_0, m_0)$ be two DSXMs having the same input alphabet, output alphabet, memory, type (Φ) and initial memory value and let l be a positive integer. Then Z' is called an l -cover DSXM of Z w.r.t. Φ if $f_Z \upharpoonright \Sigma_{\leq l} = f_{Z'} \upharpoonright \Sigma_{\leq l}$.

For a (partial) function $f : A \rightarrow B$ and $U \subseteq A$, $f \upharpoonright U$ denotes the restriction of f to U .

DEFINITION 6.2. An l -cover DSXM Z' of Z w.r.t. Φ is called *minimal* if for any l -cover DSXM Z'' of Z w.r.t. Φ , the number of states of Z' is less than or equal to the number of states of Z'' .

Note that, since any l -cover DSXM Z' of Z has the same type and initial memory value as Z , Z' is uniquely determined by its associated FA, $A_{Z'}$. Thus, Z' can be identified with $A_{Z'}$.

In the remainder of this section we show that the construction of all l -cover DSXMs of a DSXM Z whose type is input-complete and output-distinguishable can be reduced to the construction of all DFCA of $L = \mathcal{L}_{A_Z} \cap \Phi_{\leq l}$.

LEMMA 6.1. Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $Z' = (\Sigma, \Gamma, Q', M, \Phi, F', q'_0, m_0)$ be two DSXMs with type Φ , input-complete and output-distinguishable, and let l be a positive integer. Then $f_Z \upharpoonright \Sigma_{\leq l} = f_{Z'} \upharpoonright \Sigma_{\leq l}$ if and only if $\mathcal{L}_{A_Z} \cap \Phi_{\leq l} = \mathcal{L}_{A_{Z'}} \cap \Phi_{\leq l}$.

Proof. The ‘if’ implication follows directly from Definition 5.4 (note that the ‘design for test conditions’ are not necessary for this implication). We now prove the ‘only if’ implication. Assume otherwise and let $p = \phi_1 \cdots \phi_k \in \Phi^*$ such that $p \in \mathcal{L}_{A_Z} \setminus \mathcal{L}_{A_{Z'}}$. Since Φ is input-complete $\exists \sigma_1, \dots, \sigma_k \in \Sigma, \gamma_1, \dots, \gamma_k \in \Gamma, m_1, \dots, m_k \in M$ such that $\phi_i(m_{i-1}, \sigma_i) = (\gamma_i, m_i), 1 \leq i \leq k$, and $s = \sigma_1 \cdots \sigma_k \in \text{dom}(f_Z) = \text{dom}(f_{Z'})$. Since Φ is output-distinguishable, from $s \in \text{dom}(f_{Z'})$ by induction on $1 \leq i \leq k$ it follows that $\phi_i(m_{i-1}, \sigma_i) = (\gamma_i, m_i)$ and $\phi_1 \cdots \phi_i \in \mathcal{L}_{A_{Z'}}, 1 \leq i \leq k$. Thus $p \in \mathcal{L}_{A_{Z'}}$, which contradicts the original assumption. \square

Note 6.1. The ‘design for test conditions’ are essential for the ‘only if’ implication of Lemma 6.1. Without the input-completeness, some states of a DSXM may never be reached or some processing functions may never be attained. The output-distinguishability condition, on the other hand, ensures that the processing functions can be distinguished by examining the outputs produced.

For example, consider $\Sigma = \Gamma = \{a, b\}, M = \{0, 1\}, m_0 = 0$ and $\phi_1, \phi_2 : M \times \Sigma \rightarrow \Gamma \times M$ defined by:

$$\begin{aligned} \phi_1(0, a) &= (a, 0); \\ \phi_2(1, b) &= (b, 1). \end{aligned}$$

It is easy to see that $\Phi_1 = \{\phi_1, \phi_2\}$ is output-distinguishable but not input-complete. Consider $Z_1 = (\Sigma, \Gamma, \{0, 1\}, M, \Phi, F_1, 0, m_0)$ and $Z'_1 = (\Sigma, \Gamma, \{0\}, M, \Phi, F'_1, 0, m_0)$ with F_1 and F'_1 as represented in Figure 8(a) and (b), respectively, and $Z_2 = (\Sigma, \Gamma, \{0, 1\}, M, \Phi, F_2, 0, m_0)$ and $Z'_2 = (\Sigma, \Gamma, \{0, 1\}, M, \Phi, F'_2, 0, m_0)$ with F_2 and F'_2 as represented in Figure 9(a) and (b), respectively. Clearly, the state 1 of Z_1 will not be reached by any input sequence applied to this machine, so $f_{Z_1} = f_{Z'_1}$. On the other hand, the state 1 of Z_2 is reachable, but the only memory value attainable in this state is 0, which is not processed by ϕ_2 , so $f_{Z_2} = f_{Z'_2}$.

Furthermore, consider $\phi_3, \phi_4 : M \times \Sigma \rightarrow \Gamma \times M$ defined by:

$$\begin{aligned} \phi_3(m, a) &= (a, m), & m \in M; \\ \phi_4(0, a) &= (a, 0), & m \in M; \\ \phi_4(1, b) &= (b, 1), & m \in M. \end{aligned}$$

$\Psi = \{\phi_3, \phi_4\}$ and $Z_3 = (\Sigma, \Gamma, \{0, 1\}, M, \Psi, F_3, 0, m_0)$ and $Z'_3 = (\Sigma, \Gamma, \{0, 1\}, M, \Psi, F'_3, 0, m_0)$ with F_3 and F'_3 as represented in Figure 10(a) and (b), respectively. Ψ is input-complete but not output-distinguishable. Since ϕ_3 and

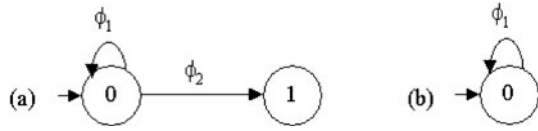


FIGURE 8. The state transition diagrams of Z_1 (a) and Z'_1 (b).

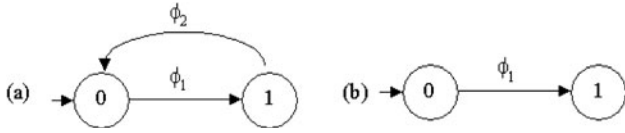


FIGURE 9. The state transition diagrams of Z_2 (a) and Z'_2 (b).

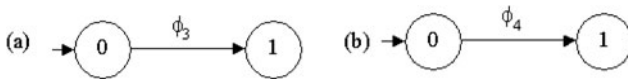


FIGURE 10. The state transition diagrams of Z_3 (a) and Z'_3 (b).

ϕ_4 can only be distinguished on the memory value 1, which is not attainable in the state 0, $f_{Z_3} = f_{Z'_3}$.

THEOREM 6.1. *Let Z be a DSXM having the type Φ input-complete and output-distinguishable, l a positive integer and $L = \mathcal{L}_{AZ} \cap \Phi_{\leq l}$. If Z' is an l -cover DSXM of Z then $A_{Z'}$ is a DFCA of L .*

Proof. Follows from Lemma 6.1, the ‘only if’ implication. \square

However, the converse is not true, as is illustrated by the following example.

EXAMPLE 6.1. Consider the DSXM Z in Example 5.1. Since $\text{dom}(\phi_1) \cap \text{dom}(\phi_2) \neq \emptyset$, Z' represented in Figure 11 is not a DSXM, so Z' is not a 2-cover DSXM of Z . However, $A_{Z'}$ is a DFCA of $L = \mathcal{L}_{AZ} \cap \Phi_{\leq 2}$.

On the other hand, the implication is valid both ways when only minimal DFCA and minimal l -cover DSXMs are considered.

THEOREM 6.2. *Let Z be a DSXM having the type Φ input-complete and output-distinguishable, l a positive integer and $L = \mathcal{L}_{AZ} \cap \Phi_{\leq l}$. Then Z' is a minimal l -cover DSXM of Z if and only if $A_{Z'}$ is a minimal DFCA of L .*

Proof. Let Z' be a minimal l -cover DSXM of Z . Then from Theorem 6.1 it follows that $A_{Z'}$ is a DFCA of L .

Conversely, let $A_{Z'}$ be a minimal DFCA of L . From Theorem 4.1 it follows that $A_{Z'}$ is isomorphic to A_L^h for some proper (maximal) SER h on L . Since Z is a DSXM, the SXM determined by A_L is also deterministic and from Definition 4.4 it follows that the SXM determined by A_L^h is also deterministic, so Z' is a DSXM. Hence, using the ‘if’ implication of Lemma 6.1, Z' is an l -cover DSXM of Z .



FIGURE 11. The non-deterministic SXM Z' .

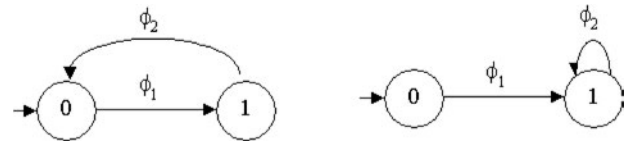


FIGURE 12. The minimal 2-cover DSXMs of Z .

Now let n_0 be the number of states of a minimal DFCA of L and k_0 be the number of states of a minimal l -cover DSXM of Z . From the two paragraphs above it follows that $n_0 \leq k_0$ and $k_0 \leq n_0$, so $n_0 = k_0$ and the required result is proven. \square

Therefore the construction of all minimal l -cover DSXMs of a DSXM Z whose type is input-complete and output-distinguishable is equivalent to the construction of all DFCA of $L = \mathcal{L}_{AZ} \cap \Phi_{\leq l}$, so the procedure given in Section 4 can be applied.

EXAMPLE 6.2. The two minimal 2-cover DSXMs of Z in Example 5.1 are represented in Figure 12.

7. CONCLUSIONS AND FURTHER WORK

In this paper a procedure for constructing all minimal cover automata of a given finite language L is proposed. An l -cover SXM of a given SXM Z is defined as a generalization of the cover automaton of a finite language L and it is proved that, in the context of SXMs that meet the ‘design for test conditions’, the problem of constructing all l -cover SXMs of Z can be reduced to the construction of all cover automata of the language comprising all sequences of length at most l accepted by the associated FA of Z .

Since the main motivation for this paper is from the domain of software testing the paper only addresses finite automata and SXMs in which all states are terminal. Further research will investigate how these results can be extended to the general case, where non-terminal states may also exist.

In practice, it is often useful to have alternative specifications, even though all of these specifications (and their implementations) provide the required functionality (i.e. in our case all DFCA or l -cover DSXMs behave identically for all sequences of length of most l). The architecture of a system can be important and one may wish to choose from among many different, functionally equivalent, specifications the one whose architecture suits best the purpose for which the system is to be constructed. For instance, a strongly

connected DFA specification is required by many finite state machine testing approaches [23]. Observe that, of the nine minimal DFCA's represented in Figure 5, only three are strongly-connected [(1), (4), (7)]. Again, only one of the two DSXMs represented in Figure 12 (the first) has a strongly connected associated FA.

Furthermore, quite often in software development the requirements are prioritized and only those with high priority can be implemented within the required cost and time constraints. In such a situation one will choose from a number of specifications that meet all the high priority requirements and the specification that meets best the low priority requirements. Implementation issues can also play a part in choosing one candidate specification rather than another.

Further work will use the theoretical results presented here to generate test sets for DFCA's and l -cover DSXMs. Traditional finite state machine [23] and SXM [19] testing methods construct test sequences whose successful application to the implementation guarantees that this implementation is equivalent to the given DFA or DSXM specification. The problem can be extrapolated to cover automata. In this case, given a specification in the form of a minimal DFCA A of a finite language L , the aim is to construct a set of input sequences whose length does not exceed l , the length of the longest sequence in L , that, when applied to any implementation A' in a class C , will detect any response of A' to input sequences of length of most l that does not conform to the response specified by A . Obviously, the set of all input sequences of length at most l can be chosen, but future work will show that, in many cases, the size of the test set may be considerably reduced. If C is the class of DFAs having the same number of states as A then the successful application of the test set to an implementation A' will ensure that the implementation under test is itself a minimal DFCA of L . This problem can also be formulated for a minimal l -cover DSXM specification.

ACKNOWLEDGEMENT

The author would like to thank the anonymous reviewers, whose useful comments have benefitted this paper.

REFERENCES

- [1] Hopcroft, J. E. and Ullman, J. D. (1979) *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, Reading, MA.
- [2] Salomaa, A. (1969) *Theory of Automata*. Pergamon Press, Oxford.
- [3] Cohen, D. I. A. (1996) *Introduction to Computer Theory* (2nd edn). John Wiley & Sons, New York.
- [4] Salomaa, K., Yu, S. and Zhuang, Q. (1994) The state complexity of some basic operations on regular languages. *Theoretical Computer Science*, **125**, 315–328.
- [5] Yu, S. (1995) *Regular Languages, Handbook of Formal Languages*. Springer Verlag.
- [6] Campeanu, C., Santean, N. and Yu, S. (1999) Minimal cover automata for finite languages. *Theoretical Computer Science*, **267**, 3–16.
- [7] Paun, A., Santean, N. and Yu, S. (2001) An $O(n^2)$ algorithm for constructing minimal cover automata for finite languages. *LNCS*, **2088**, 243–251.
- [8] Eilenberg, S. (1994) *Automata, Languages and Machines*, Vol. A. Academic Press, New York.
- [9] Holcombe, M. (1988) X-machines as a basis for dynamic system specification. *Software Engineering Journal*, **3**, 69–76.
- [10] Holcombe, M. and Ipate, F. (1998) *Correct Systems: Building a Business Process Solution*. Springer Verlag, Berlin.
- [11] Fairtlough, M., Holcombe, M., Ipate, F., Jordan, C., Laycock, G. and Duan, Z. (1995) Using an X-machine to model a video cassette recorder. *Current Issues in Electronic Modeling*, **3**, 141–161.
- [12] Kehris, E., Eleftherakis, G. and Kefalas, P. (2000) Using X-machines to model and test discrete event simulation programs. In Mastorakis, N. (ed.), *Systems and Control: Theory and Applications*, pp. 163–171. World Scientific, Athens.
- [13] Kefalas, P. and Kapeti, E. (2000) A design language and tool for X-machine specification. In Fotiadis, D. I. and Nikolopoulos, S. D. (eds), *Advances in Informatics*, pp. 134–145. World Scientific, Athens.
- [14] Ipate, F. and Holcombe, M. (1998) A method for refining and testing generalized machine specifications. *Intern. J. of Computer Math.*, **68**, 197–219.
- [15] Ipate, F. and Holcombe, M. (2002) An integrated refinement and testing method for stream X-machines. *Applicable Algebra in Engineering, Communication and Computing*, **13**, 67–91.
- [16] Barnard, J., Whitworth, J. and Woodward, M. (1996) Communicating X-machines. *Information and Software Technology*, **38**, 401–407.
- [17] Cowling, A., Georgescu, H. and Vertan, C. (2000) A structured way to use channels for communication in X-machine systems. *Formal Aspects of Computing*, **12**, 458–500.
- [18] Ipate, F. and Holcombe, M. (2002) Testing conditions for communicating stream X-machine systems. *Formal Aspects of Computing*, **13**, 431–446.
- [19] Ipate, F. and Holcombe, M. (1997) An integration testing method that is proved to find all faults. *Intern. J. Computer Math.*, **63**, 159–178.
- [20] Ipate, F. and Holcombe, M. (1998) Specification and testing using generalized machines: a presentation and a case study. *Software Testing, Verification and Reliability*, **8**, 61–81.
- [21] Ipate, F. and Holcombe, M. (2000) Generating test sequences from non-deterministic generalized stream X-machines. *Formal Aspects of Computing*, **12**, 443–458.
- [22] Ipate, F. (2003) On the minimality of stream X-machines. *Comput. J.*, **46**, 295–306.
- [23] Lee, D. and Yannakakis, M. (1996) Principles and methods of testing finite state machines – a survey. *Proc. IEEE*, **84**, 1090–1123.
- [24] Korner, H. (2003) On minimizing cover automata for finite languages in $O(n \log n)$ time. *LNCS*, **2608**, 117–127.