

On the Minimality of Stream X-machines

FLORENTIN IPATE

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

One approach to formally specifying a system is to use a form of extended finite-state machine called a *stream X-machine*. A stream X-machine is a type of X-machine 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 stream X-machine may be modelled by a finite automaton (the associated finite automaton) in which the arcs are labelled by function names (the processing functions). One important problem that may appear in the specification process is to find a ‘minimal’ (in some sense) stream X-machine that specifies the required functionality. This paper investigates the minimality issue in the context of deterministic stream X-machines and considers two types of minimality: *state-minimal* stream X-machine with respect to Φ and *minimal cover* with respect to Φ , where Φ is the set of processing functions that the machine may use.

Received 2 July 2002; revised 29 October 2002

1. INTRODUCTION

An important approach to the development of high-quality software is the use of formal methods in its specification and verification. Formal specifications and models remove the possibility of ambiguity and facilitate formal, possibly automated, analysis.

One approach to formally specifying a system is to use a form of extended finite-state machine called a *stream X-machine* [1, 2]. A stream X-machine is a type of X-machine [1, 3, 4] 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. A stream X-machine may be modelled by a finite automaton (the *associated finite automaton*) in which the arcs are labelled by function names (the *processing functions*). Thus, stream X-machines can combine the dynamic features of finite-state machines with data structures, thus sharing the benefits of both these worlds. Various case studies [1, 5, 6] have demonstrated the value of the stream X-machine as a specification method, especially for interactive systems. A tool for writing stream X-machine specifications has also been constructed [7]. The *refinement* of stream X-machines [8, 9] as well as various subclasses of stream X-machines [2, 10, 11, 12] have also been investigated. Furthermore, several models of *communicating stream X-machines* have been devised and their applicability to real applications has been demonstrated [13, 14, 15, 16, 17]. Communicating stream X-machines have also been used for modelling P-systems [18].

Another strength of using stream X-machines to specify a system is that, under certain well-defined conditions, it is

possible to produce a test set that is guaranteed to determine the correctness of an implementation [1, 19, 20]. The testing method assumes that the processing functions are correctly implemented and reduces the testing of a stream X-machine to the testing of its associated finite automaton. In practice, the correctness of the processing functions is checked by a separate process: depending on the nature of a function, it can be tested using the same method or alternative functional methods [1, 21].

Unlike the traditional extended finite-state machine testing approaches [22, 23, 24], this method does not involve the construction of the equivalent finite-state machine (whose states are the state/memory pairs of the original stream X-machine), and therefore does not rely on the finiteness of the memory and avoids the state explosion problem associated with this construction. Instead, in typical applications of the method, it is successively applied to the hierarchy of stream X-machines that are created when the processing functions are considered at lower and lower levels. Thus, testing a specific function involves considering it as a computation defined by a simpler stream X-machine and so on. Ultimately, at the lowest level, the processing functions are usually quite simple and can be tested using suitable alternative methods—for example, category partition testing [25] or a variant—or even assumed to be fault free if they are routines or objects from a library.

Furthermore, the method can guarantee the correctness of the implementation under test only if the processing functions meet some ‘design for test conditions’ *viz.* input-completeness and output-distinguishability [1, 19].

The method was first developed in the context of deterministic stream X-machines [1, 19] and then extended

to the non-deterministic case [26]. Alternative designs for test conditions have also been investigated [27]. The method in which, initially, only equivalence testing was considered, has also been extended to address conformance testing [28].

An important problem arising in the specification process is that of finding a ‘minimal’ (in some sense) specification for a required functionality. This problem has been investigated, for example, for finite-state machines and their variants (such as sequential machines) and several types of minimality have been identified: minimal finite-state machines (automata) [29] or sequential machines [3], minimal cover-automata for finite languages [30, 31], strongly minimal sequential machines [3], etc.

This paper investigates the minimality issue in the context of deterministic stream X-machines and addresses two types of minimality: *state-minimal* stream X-machine with respect to Φ and *minimal cover* with respect to Φ , where Φ is the set of processing functions that the machine may use.

A state-minimal stream X-machine with respect to Φ is the ‘smallest’ (in terms of states and arcs) deterministic stream X-machine with processing functions in Φ having a given functionality. This situation corresponds to constructing the ‘smallest’ model for a required functionality using ready-made components (the set of processing functions Φ). This is a very common situation in an object-oriented environment. The construction is also useful for the application of the stream X-machine based testing method, which assumes that the processing functions are correctly implemented; in practice this can be achieved if these are routines or objects from a library or are other previously developed pieces of software.

In some instances it might be perfectly acceptable to add extra functionality to a system, as long as the ‘core’ functionality required remains unchanged. A minimal cover with respect to Φ of a given deterministic stream X-machine is the ‘smallest’ deterministic stream X-machine with processing functions in Φ having the functionality of the original machine while possibly possessing extra functionality. The construction of a minimal cover is particularly useful when a stream X-machine is not completely specified and the missing functionality is not relevant in the context of the specified system.

The paper is structured as follows. Sections 2 and 3 introduce basic concepts of finite automata and stream X-machines, respectively. The state-minimal stream X-machine is defined in Section 4; the necessary conditions for a deterministic stream X-machine to be state-minimal are also identified. It is also established that, if the processing functions meet the two ‘design for test conditions’ (output-distinguishability and input-completeness), then state minimality is equivalent to the minimality of the associated finite automaton. Section 5 defines the concept of minimal cover and gives an algorithm for constructing all minimal covers of a stream X-machine whose processing functions satisfy the two ‘design for test conditions’. Section 6 investigates further the construction of a minimal cover for a particular class of deterministic stream X-

machines where each processing function is associated with a unique transition.

Before continuing, we introduce the notation used in the paper. For a finite alphabet A , A^* denotes the set of all finite sequences with members in A . ϵ denotes the empty sequence and $A^+ = A^* - \{\epsilon\}$. 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 a sequence $a \in A^*$, $\text{length}(a)$ denotes the number of elements of a (in particular $\text{length}(\epsilon) = 0$). For a finite set of sequences, $L \subseteq A^*$, $\text{length}(L) = \max\{\text{length}(s) \mid s \in L\}$ denotes the length of the longest sequence(s) in L .

For a sequence $a \in A^*$ we say that $b \in A^*$ is a *prefix* of a if there exists $c \in A^*$ such that $a = bc$. The set of all prefixes of a is denoted by $\text{pref}(a)$, i.e.

$$\text{pref}(a) = \{b \in A^* \mid \exists c \in A^* \text{ such that } a = bc\}.$$

For $U \subseteq A^*$, $\text{pref}(U) = \bigcup_{a \in U} \text{pref}(a)$.

For a (partial) function $f : A \rightarrow B$, $\text{dom}(f)$ denotes the domain of f , the subset of A for which f is defined. For two (partial) functions $f, g : A \rightarrow B$, we use $f \subseteq g$ to denote that $f(a) = g(a)$, $\forall a \in \text{dom}(f)$.

For a finite set A , $\text{card}(A)$ denotes the number of elements of A .

2. FINITE AUTOMATA

This section defines the finite automaton and related concepts and results to be used later in the paper.

DEFINITION 2.1. A finite automaton (FA) A is a tuple (Σ, Q, F, I, T) , where:

- 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, $I \subseteq Q$, $T \subseteq Q$.

F is usually described by a transition diagram. If $q, q' \in Q$, $\sigma \in \Sigma$ and $q' \in F(q, \sigma)$, we say that σ is an *arc* from q to q' and write $\sigma : q \rightarrow q'$.

DEFINITION 2.2. An FA is called deterministic if:

- there is one initial state, i.e.

$$I = \{q_0\};$$

- F maps each state/input pair into at most one state, i.e.

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

In what follows we will only refer to deterministic FAs with all states terminal ($T = Q$), denoted by a tuple (Σ, Q, F, q_0) .

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

- $F^*(q, \epsilon) = q$, $\forall q \in Q$;

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

DEFINITION 2.4. For $q \in Q$, the language accepted by A in q , denoted by $L_A(q)$, is defined by:

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

The language accepted by A in q_0 is simply called the language accepted by A and is denoted by L_A .

DEFINITION 2.5. A state $q \in Q$ is called accessible if $\exists s \in \Sigma^*$ with $F^*(q_0, s) = q$. 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 distinguishable if $L_A(q_1) \neq L_A(q_2)$. Otherwise q_1 and q_2 are called equivalent. A is called reduced if $\forall q_1, q_2 \in Q$, $(q_1 \neq q_2) \implies (q_1 \text{ and } q_2 \text{ are distinguishable})$.

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

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

This is a well-known result; for a proof see, for example, [3].

A morphism is a particular type of state mapping between two finite automata. This is formally defined next.

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

- $g(q_0) = q'_0$;
- $g(F(q, \sigma)) \subseteq F'(g(q), \sigma)$, $\forall q \in Q, \sigma \in \Sigma$.

The notation $g(F(q, \sigma)) \subseteq F'(g(q), \sigma)$ is used to denote that either $g(F(q, \sigma))$ is undefined or $g(F(q, \sigma)) = F'(g(q), \sigma)$.

DEFINITION 2.9. A morphism g is called proper if

$$g(F(q, \sigma)) = F'(g(q), \sigma), \forall q \in Q, \sigma \in \Sigma.$$

DEFINITION 2.10. A morphism is called an isomorphism if it is proper and bijective.

Note 2.1. If a morphism g from A and A' is bijective then either g is an isomorphism or A can be obtained from A' by removing one or more arcs. This observation will be used later in the paper.

LEMMA 2.1. If $g : Q \rightarrow Q'$ is a morphism from A to A' then $L_A(q) \subseteq L_{A'}(g(q))$, $\forall q \in Q$. In particular, $L_A \subseteq L_{A'}$. If g is proper then $L_A(q) = L_{A'}(g(q))$, $\forall q \in Q$. In particular, $L_A = L_{A'}$.

Proof. By induction on the length of $s \in \Sigma^*$ it follows that $g(F^*(q, s)) \subseteq F'^*(g(q), s)$, $\forall q \in Q, s \in \Sigma^*$. Hence $L_A(q) \subseteq L_{A'}(g(q))$. If g is proper then $g(F^*(q, s)) = F'^*(g(q), s)$, $\forall q \in Q, s \in \Sigma^*$. \square

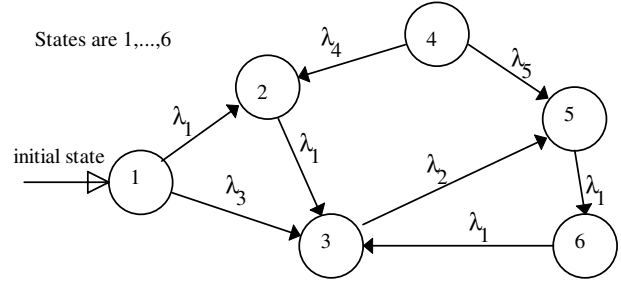


FIGURE 1. The state transition diagram of an X-machine.

THEOREM 2.2. For two minimal deterministic FAs A and A' , $L_A = L_{A'}$ if and only if A and A' are isomorphic.

This is a well-known result; for a proof see, for example, [3]. Techniques for constructing the minimal FA that accepts a given language also exist; for more details see, for example, [3].

3. STREAM X-MACHINES AND STREAM FUNCTIONS

In this section the stream X-machine and other basic concepts related to it are defined.

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 1, 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 then 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 there is only one possible function defined (that is 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 stream X-machines and are defined formally next. The basic idea is that the machine has some internal memory, M , and the stream of inputs determine, 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 3.1. A stream X-Machine (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;
- M is a (possibly) infinite set called memory;
- Q is the finite set of states;
- Φ is the type of Z , a finite set of non-empty basic processing functions that the machine can use, having the form

$$\phi : M \times \Sigma \longrightarrow \Gamma \times M;$$

- F is the (partial) next state function,

$$F : Q \times \Phi \longrightarrow 2^Q;$$

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 $\phi : M \times \Sigma \longrightarrow \Gamma \times M$, 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 a finite automaton 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 . Analogously to finite automata, if for $q, q' \in Q$, $\phi \in \Phi$, $F(q, \phi) = q'$, then ϕ is called an *arc* of Z from q to q' , denoted $\phi : q \rightarrow q'$.

DEFINITION 3.2. Given a sequence $p \in \Phi^*$, p induces the (partial) function

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

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 stream X-machine 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 3.3. An SXM Z is called deterministic if the following hold.

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

$$I = \{q_0\};$$

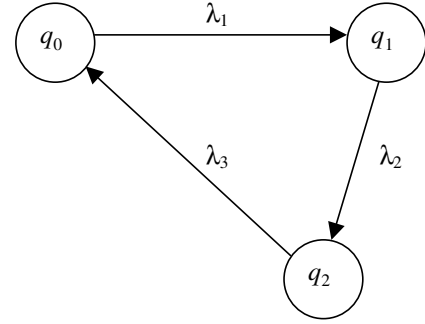


FIGURE 2. The associated FA of Z .

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

$$F : Q \times \Phi \longrightarrow Q;$$

- Any two distinct processing functions that label arcs emerging from the same state have disjoint domains, i.e.

$$\forall \phi_1, \phi_2 \in \Phi \text{ if } \exists q \in Q \text{ with } (q, \phi_1), (q, \phi_2) \in \text{dom}(F) \text{ then } \phi_1 = \phi_2 \text{ or } \text{dom}(\phi_1) \cap \text{dom}(\phi_2) = \emptyset.$$

Since stream X-machine specifications are often used as a basis for testing, it is normal to assume that every state is a terminal state, i.e. $T = Q$. Furthermore, this is not normally a restriction when considering interactive systems.

Thus, in what follows, we will refer to deterministic SXMs with all states 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 3.1. A deterministic SXM $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ with $\Sigma = \{a, b\}$, $\Gamma = \{x, y, z, w\}$, $Q = \{q_0, q_1, q_2\}$, $M = \{0, 1\}$, $m_0 = 0$, $\Phi = \{\phi_1, \phi_2, \phi_3\}$, F as represented in Figure 2 and $\phi_1, \phi_2, \phi_3 : M \times \Sigma \longrightarrow \Gamma \times M$ defined by:

$$\begin{aligned} \phi_1(m, a) &= (x, m), & m \in M; \\ \phi_2(0, b) &= (z, 0), & \phi_2(1, b) = (w, 1); \\ \phi_3(m, a) &= (y, 1 - m), & m \in M; \end{aligned}$$

will be used in illustrations later in the paper.

DEFINITION 3.4. A state $q \in Q$ is called *reachable* if $\exists p \in \Phi^*$, $m \in M$, $s \in \Sigma^*$, $g \in \Gamma^*$ such that $F^*(q_0, p) = q$ and $|p|(m_0, s) = (g, m)$.

DEFINITION 3.5. Given a state $q \in Q$, a memory value $m \in M$ is called *attainable* in q if $\exists p \in \Phi^*$, $s \in \Sigma^*$, $g \in \Gamma^*$ such that $F^*(q_0, p) = q$ and $|p|(m_0, s) = (g, m)$. The set of all memory values attainable in q make up the set $M\text{Attain}_Z(q)$.

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 3.6. Given a deterministic SXM Z , the (partial) function $f_Z : \Sigma^* \rightarrow \Gamma^*$ defined by

$$f_Z(s) = g \text{ if } \exists p \in \Phi^*, m \in M \\ \text{such that } (q_0, p) \in \text{dom}(F^*) \text{ and } |p|(m_0, s) = (g, m)$$

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. This paper will, however, deal only with the deterministic case.

DEFINITION 3.7. A (partial) function $f : \Sigma^* \rightarrow \Gamma^*$ is called length preserving if $\text{length}(s) = \text{length}(f(s))$, $\forall s \in \text{dom}(f)$.

DEFINITION 3.8. A (partial) function $f : \Sigma^* \rightarrow \Gamma^*$ is called segment preserving if the following holds: $\forall s, t \in \Sigma^*$, $(st \in \text{dom}(f)) \implies (s \in \text{dom}(f) \text{ and } \exists v \in \Gamma^* \text{ such that } f(st) = f(s)v)$.

DEFINITION 3.9. A (partial) function $f : \Sigma^* \rightarrow \Gamma^*$ is called a stream function (S function) if f is length preserving and segment preserving.

THEOREM 3.1 ([1]). If Z is an SXM then f_Z is an S function.

EXAMPLE 3.2. For Z as in Example 3.1, $f_Z : \Sigma^* \rightarrow \Gamma^*$ is the S function induced by

$$f_Z((abaaba)^n) = (xzyxwy)^n, \quad n \geq 0$$

(i.e. $f_Z(s) = t$, $\forall s \in \text{pref}(\{(abaaba)^n \mid n \geq 0\})$, $t \in \text{pref}(\{(xzyxwy)^n \mid n \geq 0\})$ with $\text{length}(s) = \text{length}(t)$).

A more precise characterization of the functions computed by SXMs will obviously depend on the nature of the processing functions and memory used (different subclasses of SXMs may be defined such as ‘finite’ SXMs, ‘pushdown’ SXMs, etc., see for example [2, 3]). This is not, however, relevant in the context of this paper.

4. STATE-MINIMAL SXMS

This section defines the concept of *state-minimal* SXM. A state-minimal SXM with respect to a type Φ is the ‘smallest’ (in terms of states and arcs) deterministic SXM Z with type Φ that computes the function f_Z . This situation corresponds to constructing the ‘smallest’ model for a given functionality using ready-made components (the set of processing functions Φ).

We show that, in general, the state-minimality of a deterministic SXM cannot be reduced to the minimality

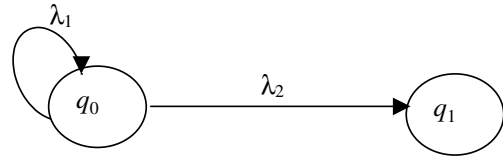


FIGURE 3. The associated FA of Z .

of its associated FA and identify necessary conditions for a deterministic SXM to be state-minimal. On the other hand, we show that if the type of the SXM satisfies the two ‘design for test conditions’ (output-distinguishability and input-completeness) then state-minimality is equivalent to the minimality of the associated FA.

DEFINITION 4.1. Two deterministic SXMs are called testing compatible if they have identical input alphabets, output alphabets, memory sets, types and initial memory values.

DEFINITION 4.2. A deterministic SXM Z is called state-minimal with respect to Φ if the following hold:

- if Z' is a deterministic SXM that is testing compatible with Z and $f_Z = f_{Z'}$ then the number of states of Z' is at least equal to the number of states Z .
- by removing one or more arcs from Z the function computed will change.

Note 4.1. The second condition above can be formalized in terms of FA morphisms:

- if Z' is a deterministic SXM that is testing compatible with Z such that $f_Z = f_{Z'}$ and there exists a bijective morphism from $A_{Z'}$ to A_Z then $A_{Z'}$ and A_Z are isomorphic.

We now identify necessary conditions for a deterministic SXM to be state minimal.

LEMMA 4.1. If Z is state-minimal with respect to Φ then A_Z is minimal.

Proof. Otherwise, the associated FA can be minimized without affecting the function computed by the machine. \square

However, the converse may not be true if, for example, one or more states of the automaton are not reachable, as is illustrated by the following example.

EXAMPLE 4.1. Consider the deterministic SXM $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ with $\Sigma = \{a, b\}$, $\Gamma = \{x, y\}$, $Q = \{q_0, q_1\}$, $M = \{0, 1\}$, $m_0 = 0$, $\Phi = \{\phi_1, \phi_2\}$, F as represented in Figure 3 and $\phi_1, \phi_2 : M \times \Sigma \rightarrow \Gamma \times M$ defined by:

$$\phi_1(0, a) = (x, 0); \\ \phi_2(1, b) = (y, 1).$$

It is easy to see that A_Z is minimal. However, q_1 is not reachable in Z since ϕ_1 can only produce the memory

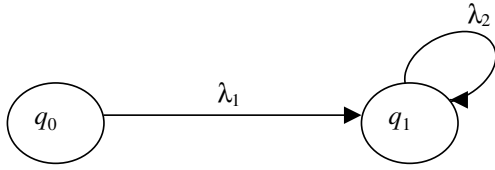


FIGURE 4. The associated FA of Z .

value 0 and ϕ_2 can only process the memory value 1. Thus, Z is not state-minimal since q_1 may be removed without affecting the function computed.

We then have the following result.

LEMMA 4.2. *If Z is state-minimal with respect to Φ then each state is reachable.*

Even if each state of the machine is reachable, the minimality of the associated automaton does not guarantee that the machine is state-minimal, as is illustrated by the following example.

EXAMPLE 4.2. Consider the deterministic SXM $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ with $\Sigma = \{a, b\}$, $\Gamma = \{x, y\}$, $Q = \{q_0, q_1\}$, $M = \{0, 1\}$, $m_0 = 0$, $\Phi = \{\phi_1, \phi_2\}$, F as represented in Figure 4 and $\phi_1, \phi_2 : M \times \Sigma \rightarrow \Gamma \times M$ defined as in Example 4.1.

Observe that A_Z is minimal and that q_0 and q_1 are reachable. However, since $MAttain(q_1) = \{0\}$ and $\text{dom}(\phi_2) \cap (\{0\} \times \Sigma) = \emptyset$, ϕ_2 may be removed from Figure 4 without changing the function computed by the machine.

Then we have the following result.

LEMMA 4.3. *If Z is state-minimal with respect to Φ then $\forall q \in Q, \phi \in \Phi, ((q, \phi) \in \text{dom}(F)) \implies (\text{dom}(\phi) \cap (MAttain(q) \times \Sigma) \neq \emptyset)$.*

On the other hand, state-minimality of a deterministic SXM can be reduced to the minimality of the associated automaton if the type Φ meets certain conditions, as defined next.

DEFINITION 4.3. Φ is called output-distinguishable if $\forall \phi_1, \phi_2 \in \Phi, (\exists m, m_1, m_2 \in M, \sigma \in \Sigma, \gamma \in \Gamma \text{ with } \phi_1(m, \sigma) = (\gamma, m_1) \text{ and } \phi_2(m, \sigma) = (\gamma, m_2)) \implies (\phi_1 = \phi_2)$.

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.

DEFINITION 4.4. 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 all of the accessible states in A_Z are reachable in Z . Also, it precludes the situation in which there exists an arc ϕ from q with $\text{dom}(\phi) \cap (MAttain(q) \times \Sigma) = \emptyset$.

These two conditions (output-distinguishability and input-completeness) are generally known as 'design for test conditions' [1, 19].

EXAMPLE 4.3. For Z as in Example 3.1, Φ is output-distinguishable and input-complete whereas in Example 4.1 and 4.2 Φ is output-distinguishable but not input-complete.

LEMMA 4.4. *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 deterministic testing compatible SXMs with type Φ output-distinguishable and input-complete. If $f_Z = f_{Z'}$ then $L_{A_Z} = L_{A_{Z'}}$.*

Proof. Assume otherwise and let $p = \phi_1 \dots \phi_k \in \Phi^*$ such that $p \in L_{A_Z} \setminus L_{A_{Z'}}$. 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 $\phi_i(m_{i-1}, \sigma_i) = (\gamma_i, m_i), \forall 1 \leq i \leq k$, and $s = \sigma_1 \dots \sigma_k \in \text{dom}(f_Z) = \text{dom}(f_{Z'})$. 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 $(q'_0, \phi_1 \dots \phi_i) \in \text{dom}(F'^*)$, $\forall 1 \leq i \leq k$. Thus $p \in L_{A_{Z'}}$, which contradicts the original assumption. \square

THEOREM 4.1. *Let Z be a deterministic SXM with Φ output-distinguishable and input-complete. If the associated automaton A_Z is minimal then Z is state-minimal with respect to Φ .*

Proof. Let Z' be a state-minimal SXM that computes f_Z . Then $A_{Z'}$ is minimal. From Lemma 4.4 and Theorem 2.2 it follows that A_Z and $A_{Z'}$ are isomorphic. Hence Z is state minimal with respect to Φ . \square

5. MINIMAL SXM COVERS

Let us assume that we have an SXM specification, say Z , of a system. In some instances it might be acceptable to add extra functionality to the system, as long as the 'core' functionality specified by f_Z remains unchanged. Therefore, any machine Z' that satisfies $f_Z \subseteq f_{Z'}$ will also be an acceptable specification of the same system.

In this context, a natural question that arises is how we can construct such a Z' with minimum number of states and arcs. This construction is particularly useful when Z is not completely specified and the missing functionality is not relevant for the specified system.

In what follows we shall address this problem for deterministic SXMs whose type meets the 'design for test conditions'. Let us formalize the problem first.

DEFINITION 5.1. *Let Z and Z' be two testing compatible deterministic SXMs with type Φ . Then Z' is called a cover of Z with respect to Φ if $f_Z \subseteq f_{Z'}$.*

DEFINITION 5.2. *Z' is called a minimal cover of Z with respect to Φ if the following hold:*

- Z' is a cover of Z with respect to Φ ;
- if Z'' is a cover of Z with respect to Φ then the number of states of Z'' is at least equal to the number of states of Z' ;
- if Z'' is a deterministic SXM obtained by removing one or more arcs from Z' then Z'' is no longer a cover of Z .

Note 5.1. Similar to Definition 4.2, the last condition can be formalized in terms of FA morphisms.

In the remainder of this section we show how the set of all minimal covers can be constructed for the case where Φ is output-distinguishable and input-complete.

DEFINITION 5.3. *Given two deterministic testing compatible SXMs, $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $Z' = (\Sigma, \Gamma, Q', M, \Phi, F', q'_0, m_0)$, we write $A_{Z'} \geq A_Z$ if there exists a function $g : Q \rightarrow Q'$ with:*

- $g(q_0) = q'_0$;
- $L_{A_Z}(q) \subseteq L_{A_{Z'}}(g(q)), \forall q \in Q$.

Obviously, if $A_{Z'} \geq A_Z$ then $L_{A_Z} \subseteq L_{A_{Z'}}$. The converse is also true if A_Z is accessible, as is shown next.

LEMMA 5.1. *If $L_{A_Z} \subseteq L_{A_{Z'}}$ and A_Z is accessible then $A_{Z'} \geq A_Z$.*

Proof. Define g as follows:

- $g(q_0) = q'_0$;
- for $q \in Q - \{q_0\}$ let $p_q \in \Phi^*$ such that $F^*(q_0, p_q) = q$ (such a p_q exists since A_Z is accessible). Then take $g(q) = F'^*(q'_0, p_q)$. Note that $F'^*(q'_0, p_q)$ is defined since $L_{A_Z} \subseteq L_{A_{Z'}}$.

It is easy to verify that $L_{A_Z}(q) \subseteq L_{A_{Z'}}(g(q)), \forall q \in Q$. Obviously, g defined above might not be unique. \square

If Z and Z' are two testing compatible deterministic SXMs and $A_{Z'} \geq A_Z$ then $f_Z \subseteq f_{Z'}$. The converse is also true if Φ is output-distinguishable and input-complete and A_Z is accessible.

LEMMA 5.2. *Let Z and Z' be two testing compatible deterministic SXMs with type Φ which is output-distinguishable and input-complete. If $f_Z \subseteq f_{Z'}$ and A_Z is accessible then $A_{Z'} \geq A_Z$.*

Proof. Similar to the proof of Lemma 4.4 we have $L_{A_Z} \subseteq L_{A_{Z'}}$. Then from Lemma 5.1 it follows that $A_{Z'} \geq A_Z$. \square

Therefore, if Z is a deterministic SXM with Φ output-distinguishable and input-complete and its associated FA, A_Z , is minimal, then the problem of finding all minimal covers of Z can be reduced to finding all deterministic SXMs Z' such that:

1. $A_{Z'} \geq A_Z$;
2. if Z'' is a deterministic SXM such that $A_{Z''} \geq A_Z$, then $A_{Z''}$ has at least as many states as $A_{Z'}$;
3. if A' is an FA obtained from $A_{Z'}$ by removing one or more arcs, then $A' \geq A_Z$ does not hold.

We now show how the set of all minimal covers can be effectively constructed using special kinds of decompositions of the state set of Z , as defined next.

DEFINITION 5.4. *A set $H = \{H_i\}_{i \in I}$ is called a decomposition of a set X if $H_i \neq \emptyset, \forall i \in I$ and $\bigcup_{i \in I} H_i = X$. If $H_i \cap H_j = \emptyset, \forall i, j \in I$ with $i \neq j$ then H is called a partition of X .*

DEFINITION 5.5. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM. Then a set $H = \{H_i\}_{i \in I}$ is called an admissible decomposition of Q if:*

- H is a decomposition of Q ;
- $\forall i \in I, \phi \in \Phi, \exists j \in I$ such that $\forall q \in H_i, ((q, \phi) \in \text{dom}(F)) \implies (F(q, \phi) \in H_j)$.

DEFINITION 5.6. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM. Then two states $q, q' \in Q$ are called domain-compatible if $\forall \phi, \phi' \in \Phi$ if $(q, \phi), (q', \phi') \in \text{dom}(F)$, then either $\phi = \phi'$ or $\text{dom}(\phi) \cap \text{dom}(\phi') = \emptyset$. Otherwise, q and q' are called domain-incompatible.*

DEFINITION 5.7. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM. Then a set $H = \{H_i\}_{i \in I}$ is called a domain-consistent decomposition of Q if:*

- H is a decomposition of Q ;
- $\forall i \in I, (q, q' \in H_i) \implies (q \text{ and } q' \text{ are domain-compatible})$.

The following lemma establishes that if $A_{Z'} \geq A_Z$ then Z' determines an admissible and domain-consistent decomposition of the state set of Z .

LEMMA 5.3. *Let Z and Z' be two deterministic SXMs such that $A_{Z'} \geq A_Z$. Let $h : Q' \rightarrow 2^Q$ defined by*

$$h(q') = \{q \in Q \mid L_Z(q) \subseteq L_{Z'}(q')\}.$$

Then the set $H = \{H_{q'}\}_{q' \in I}$, where $I = \{q' \in Q' \mid h(q') \neq \emptyset\}$ and $H_{q'} = h(q'), \forall q' \in I$, is an admissible and domain-consistent decomposition of Q' .

Proof. Let $q' \in I$ and $q_1, q_2 \in h(q')$. Then $L_Z(q_1) \subseteq L_{Z'}(q')$ and $L_Z(q_2) \subseteq L_{Z'}(q')$. Let $\phi_1, \phi_2 \in \Phi, \phi_1 \neq \phi_2$ with $(q_1, \phi_1), (q_2, \phi_2) \in \text{dom}(F)$. Then $(q', \phi_1), (q', \phi_2) \in \text{dom}(F')$ so $\text{dom}(\phi_1) \cap \text{dom}(\phi_2) \neq \emptyset$ since Z' is deterministic. Hence H is domain-compatible.

H is admissible since $\forall q' \in Q', \phi \in \Phi, q \in Q, (q \in h(q') \text{ and } (q, \phi) \in \text{dom}(F)) \implies (F(q, \phi) \in h(F'(q', \phi)))$. \square

Conversely, an admissible and domain-consistent decomposition of the state set of Z determines at least one machine Z' whose associated FA satisfies $A_{Z'} \geq A_Z$.

DEFINITION 5.8. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM and $H = \{H_i\}_{i \in I}$ be an admissible and domain-consistent decomposition of Q . Then Z/H denotes the set of all deterministic SXMs $Z' = (\Sigma, \Gamma, Q', M, \Phi, F', q'_0, m_0)$ that meet the following conditions:*

- $Q' = \{H_i\}_{i \in I}$;
- $F'(H_i, \phi)$ is as follows: $\forall i \in I, \phi \in \Phi,$
 - if $\exists q \in H_i$ with $(q, \phi) \in \text{dom}(F)$ then $F'(H_i, \phi) = H_j$, where j is such that $F(q, \phi) \in H_j, \forall q \in H_i,$
 - otherwise $F'(H_i, \phi)$ is undefined;
- $q_0 \in q'_0$.

Note 5.2. Since H is admissible the set Z/H is not empty (i.e. there exists at least one F' as above). Since Z is deterministic and H is domain-consistent, any element of Z/H is also a *deterministic SXM*. Indeed, let $i \in I$ and $\phi_1, \phi_2 \in \Phi$ with $\phi_1 \neq \phi_2$, $(H_i, \phi_1), (H_i, \phi_2) \in \text{dom}(F')$. Then $\exists q, q' \in H_i$ with $(q, \phi_1), (q', \phi_2) \in \text{dom}(F)$. Since q and q' are domain-compatible it follows that $\text{dom}(\phi_1) \cap \text{dom}(\phi_2) = \emptyset$.

Note 5.3. Z/H will have more than one element if $\exists i, j, k \in I$ and $\exists \phi \in \Phi$ with $j \neq k$, $F(H_i, \phi) \subseteq H_j$ and $F(H_i, \phi) \subseteq H_k$ or $\exists i, j \in I$ with $i \neq j$ and $q_0 \in H_i \cap H_j$. If H is a partition, then Z/H will contain exactly one element.

LEMMA 5.4. *Let Z be a deterministic SXM and $Z' \in Z/H$. Then $A_{Z'} \geq A_Z$.*

Proof. We can define at least one function $g : Q \rightarrow Q'$ by $g(q) = H_i$, where i is such that $q \in H_i$. Then it is easy to verify that $L_Z(q) \subseteq L_{Z'}(g(q))$, $\forall q \in Q$. \square

COROLLARY 5.1. *Let Z a deterministic SXM and $Z' \in Z/H$. Then Z' is a cover of Z .*

Proof. Follows from Lemma 5.4. \square

THEOREM 5.1. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM with Φ output-distinguishable and input-complete and A_Z minimal and let $Z' = (\Sigma, \Gamma, Q', M, \Phi, F', q'_0, m_0)$ be a minimal cover of Z . Then there exists $H = \{H_i\}_{i \in I}$ an admissible and domain-consistent decomposition of Q and $Z'' \in Z/H$ such that $A_{Z'}$ and $A_{Z''}$ are isomorphic.*

Proof. Since A_Z is minimal, A_Z is accessible so, from Lemma 5.2, it follows that $A_{Z'} \geq A_Z$. Let $h : Q' \rightarrow 2^Q$ defined by $h(q') = \{q \in Q \mid L_Z(q) \subseteq L_{Z'}(q')\}$. Then the set $H = \{H_{q'}\}_{q' \in I}$, where $I = \{q' \in Q' \mid h(q') \neq \emptyset\}$ and $H_{q'} = h(q')$, $\forall q' \in I$, is an admissible and domain-consistent decomposition of Q' . From Lemma 5.4 it follows that any element of Z/H is a cover of Z . Since Z' is a minimal cover, $\text{card}(Q') \leq \text{card}(H)$. Now, from the construction of the set H , we have $\text{card}(H) \leq \text{card}(Q')$. Hence $\text{card}(H) = \text{card}(Q')$ and the function $h_r : Q' \rightarrow H$ defined by $h_r(q') = h(q')$, $\forall q' \in Q'$, is bijective. Let $k = h_r^{-1}$, $k : H \rightarrow Q'$.

Then we construct $Z'' = (\Sigma, \Gamma, Q'', M, \Phi, F'', q''_0, m_0)$ as follows:

- $Q'' = H$;
- F'' is defined by: $\forall H_i \in H, \phi \in \Phi$,
 - $F''(H_i, \phi) = h(F'(k(H_i), \phi))$ if $\exists q \in H_i$ such that $(q, \phi) \in \text{dom}(F)$,
 - $F''(H_i, \phi)$ is undefined, otherwise;
- $q''_0 = h(q'_0)$.

It is easy to verify that $Z'' \in Z/H$. Also, $F''(H_i, \phi) \subseteq h(F'(k(H_i), \phi))$. Since $k \circ h = 1_H$, it follows that $k(F''(H_i, \phi)) \subseteq F'(k(H_i), \phi)$. Hence k is a bijective morphism from $A_{Z''}$ to $A_{Z'}$. Since Z' is a minimal cover

of Z with respect to Φ , k is an isomorphism. Hence $A_{Z'}$ and $A_{Z''}$ are isomorphic. \square

Therefore, if $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ is a deterministic SXM with Φ output-distinguishable and input-complete and we denote by \mathcal{H} the set of all admissible and output-consistent decompositions of Q with minimum number of elements then all minimal covers of Z will be among the elements of $\Xi = \{Z' \in Z/H \mid H \in \mathcal{H}\}$.

An algorithm for determining all minimal covers of Z is given in what follows. It consists of the following steps.

1. For each pair of processing functions, determine whether they have disjoint domains or not.
2. Construct all the admissible and domain-consistent decompositions H of Q with minimum number of elements (let this number be n_0). Since $n_0 \leq n$, it is clear that there exists only a finite number of such decompositions.
3. For each such H , construct all $Z' \in Z/H$. The set of all such Z' is denoted by Ξ .
4. Remove from Ξ all the machines Z' for which $\exists Z'' \in \Xi$ such that $A_{Z''}$ can be obtained by removing one or more arcs from $A_{Z'}$. The remaining elements of Ξ are all minimal covers of Z .

Note that, of all the steps of the above algorithm, the first is the only one that uses the processing functions; if the complexity of any processing function is at most C then the complexity of this step will be at most proportional to $C \cdot k^2$, where $k = \text{card}(\Phi)$. The other three steps do not use the processing functions at all so the remainder of the algorithm reduces to an algorithm on a finite automaton (the associated FA). In the worst case, all decompositions of Q having n_0 elements will be constructed. If a backtracking procedure is used that places, in turn, the elements of Q in one or more of the n_0 parts, then the complexity of this procedure will be $2^{n_0} + 2^{2 \cdot n_0} + \dots + 2^{(n-1) \cdot n_0} \approx 2^{n_0 \cdot n}$, where $n = \text{card}(Q)$. In practice, however, the number of decompositions constructed is significantly smaller since many are ruled out by the conditions required (domain-consistency and admissibility). A more in-depth analysis of this subject will, however, require further investigation.

EXAMPLE 5.1. Consider Z in Example 3.1. Then q_0 and q_1 are domain-compatible, q_1 and q_2 are domain-compatible, but q_0 and q_2 are not domain-compatible. Thus, there are three admissible and domain-consistent decompositions of Q with minimum number of elements (two):

$$\begin{aligned} H_1 &= \{\{q_0\}, \{q_1, q_2\}\}, \\ H_2 &= \{\{q_0, q_1\}, \{q_2\}\}, \\ H_3 &= \{\{q_0, q_1\}, \{q_1, q_2\}\}. \end{aligned}$$

Then $Z/H_1 = \{Z_1\}$, $Z/H_2 = \{Z_2\}$, $Z/H_3 = \{Z_3, Z_4\}$, where

$$Z_1 = (\Sigma, \Gamma, H_1, M, \Phi, F_1, \{q_0\}, m_0) \text{ with } F_1 \text{ represented in Figure 5a,}$$

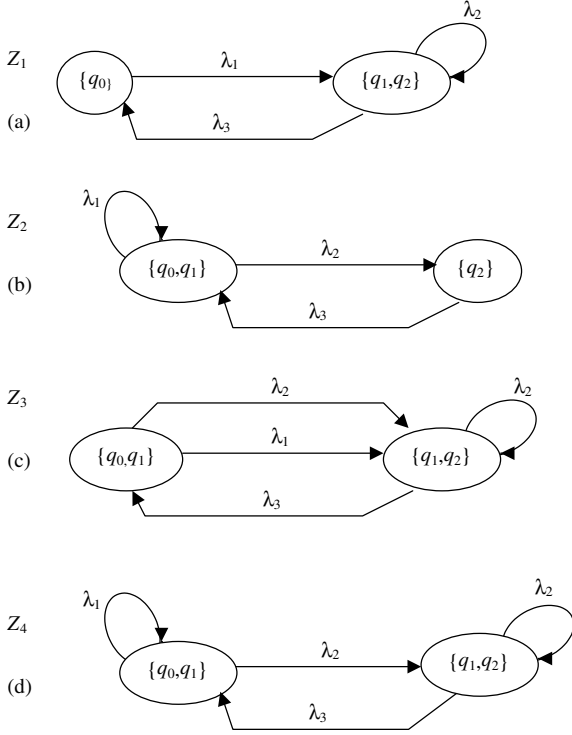


FIGURE 5. The associated FAs of Z_1 (a), Z_2 (b), Z_3 (c) and Z_4 (d).

$$\begin{aligned} Z_2 &= (\Sigma, \Gamma, H_2, M, \Phi, F_2, \{q_0, q_1\}, m_0) \\ &\text{with } F_2 \text{ represented in Figure 5b,} \\ Z_3 &= (\Sigma, \Gamma, H_3, M, \Phi, F_3, \{q_0, q_1\}, m_0) \\ &\text{with } F_3 \text{ represented in Figure 5c,} \\ Z_4 &= (\Sigma, \Gamma, H_3, M, \Phi, F_4, \{q_0, q_1\}, m_0) \\ &\text{with } F_4 \text{ represented in Figure 5d.} \end{aligned}$$

It is easy to verify that Z_1 and Z_2 can be obtained by removing one arc from Z_3 and Z_4 , respectively. Thus, the minimal covers of Z are Z_1 and Z_2 .

$f_{Z_1}, f_{Z_2} : \Sigma^* \rightarrow \Gamma^*$ are the S functions induced by:

$$\begin{aligned} f_{Z_1}((ab^i aab^j a)^n) &= (xz^i yxw^j y)^n, \quad i, j, n \geq 0, \\ f_{Z_2}((a^i baa^j ba)^n) &= (x^i zy x^j wy)^n, \quad i, j, n \geq 0. \end{aligned}$$

Clearly, $f_Z \subseteq f_{Z_1}$ and $f_Z \subseteq f_{Z_2}$.

If the type Φ is not output-distinguishable or input-complete, then this algorithm can still be used to find covers of a deterministic SXM Z with number of states less than or equal to the number of states of Z . However, in this case the machines obtained are not guaranteed to be minimal covers.

The following results are the particular forms of Theorem 5.1 for the cases where any two states are domain-compatible and domain-incompatible, respectively.

COROLLARY 5.2. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM with Φ output-distinguishable and input-complete and A_Z minimal. If any two distinct states of Z are domain-incompatible then Z is its own unique (up to an FA isomorphism) minimal covering.*

Proof. $H_Q = \{\{q\} \mid q \in Q\}$ is the only admissible and domain-consistent decomposition of Q and $Z/H_Q = \{Z\}$. \square

COROLLARY 5.3. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM with Φ output-distinguishable and input-complete and A_Z minimal and let $Z_0 = (\Sigma, \Gamma, \{0\}, M, \Phi, F_0, 0, m_0)$ with $F_0 : \{0\} \times \Phi \rightarrow \{0\}$ defined by $F_0(0, \phi) = 0$ if $\exists q \in Q$ such that $(q, \phi) \in \text{dom}(F)$. If any two distinct states of Z are domain-compatible then Z_0 is the unique (up to an FA isomorphism) minimal covering of Z .*

Proof. $H_0 = \{Q\}$ is an admissible and domain-consistent decomposition of Q and $Z/H_0 = \{Z_0\}$. \square

That is, a SXM whose states are all domain-compatible can be covered by a one-state SXM using the same type.

6. MINIMAL COVERS OF 1-COMPLEX SXMS

In many case studies investigated in previous papers, each processing function appears only once in the associated FA of an SXM specification; see, for example, the specification of an estimator in [1] or the specification of a calculator in [32].

This section investigates further the construction of a minimal covering for this particular type of SXM.

DEFINITION 6.1. *A deterministic SXM $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ is called n -complex, $n \geq 1$, if $\max_{\phi \in \Phi} \text{card}(\{q \in Q \mid (q, \phi) \in \text{dom}(F)\}) = n$.*

LEMMA 6.1. *If $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ is a deterministic 1-complex SXM then any decomposition of Q is admissible.*

Proof. Follows from Definition 5.5 and Definition 6.1. \square

DEFINITION 6.2. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM. Then $X \subseteq Q$ is called a domain-similar set of Z if $\forall q_1, q_2 \in X$, q_1 and q_2 are domain-compatible.*

DEFINITION 6.3. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM. Then $X \subseteq Q$ is called a domain-dissimilar set of Z if $\forall q_1, q_2 \in X$, $(q_1 \neq q_2) \implies (q_1$ and q_2 are domain-incompatible).*

DEFINITION 6.4. *A domain-dissimilar set X of Z is called a maximal domain-dissimilar set of Z if, for any domain-dissimilar set Y of Z , $\text{card}(Y) \leq \text{card}(X)$.*

DEFINITION 6.5. *A domain-dissimilar set $X = \{q_1, \dots, q_k\}$ of Z is called a canonical domain-dissimilar set of Z if there exists a domain-consistent partition $H = \{H_1, \dots, H_k\}$ of Q such that $q_i \in H_i$, $\forall 1 \leq i \leq k$. H is called a partition induced by X on Z .*

Note 6.1. H may not be unique. For Z as in Example 3.1 and Example 5.1, $\{q_0, q_2\}$ is a canonical domain-dissimilar set of Z and may induce two domain-consistent partitions: H_1 and H_2 .

Note 6.2. A canonical domain-dissimilar set of Z may not always exist. For example, let Z have state set $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$ and the following domain-incompatible pairs of states: q_1 and q_2 , q_1 and q_4 , q_2 and q_3 , q_2 and q_5 , q_4 and q_6 , q_5 and q_6 (with all remaining pairs being domain-compatible). It can be observed that there is no canonical domain-dissimilar set of Z .

THEOREM 6.1. *A canonical domain-dissimilar set of Z is a maximal domain-dissimilar set of Z .*

Proof. Let $X = \{q_1, \dots, q_k\}$ be a canonical domain-dissimilar set of Z and $H = \{H_1, \dots, H_k\}$ a domain-consistent partition induced by X on Z . Let $Y = \{q'_1, \dots, q'_n\}$ be an arbitrary domain-dissimilar set of Z . We assume that $n > k$. Then there exist q'_i and q'_j , $i \neq j$, such that $q'_i, q'_j \in H_m$ for some m , $1 \leq m \leq k$, which contradicts the fact that H_m is domain-similar. Hence X is a maximal domain-dissimilar set. \square

That is, all canonical domain-dissimilar sets of Z can be found among the maximal domain-dissimilar sets of Z . The converse may not always be true since, for example, a canonical domain-dissimilar set of Z may not always exist (unlike a maximal domain-dissimilar set).

LEMMA 6.2. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic 1-complex SXM with Φ output-distinguishable and input-complete and A_Z minimal, and let n_0 be the number of states of a minimal cover of Z . If X is a canonical domain-dissimilar set of Z then $\text{card}(X) = n_0$.*

Proof. Let X be a canonical domain-dissimilar set of Z and K a partition induced by X on Z . Then K is a domain-consistent decomposition of Q , so all elements of Z/K are coverings of Z . Hence $n_0 \leq \text{card}(X)$. If Z' is a minimal cover of Z then there exists a domain compatible decomposition H of Q such that $Z' \in Z/H$ and $\text{card}(H) = n_0$. Assume that $\text{card}(H) < \text{card}(X)$. Then $\exists H_1 \in H$, $q_1, q_2 \in X$ such that $q_1 \in H_1$ and $q_2 \in H_1$ and hence q_1 and q_2 are domain compatible, which contradicts the assumption that X is a domain-dissimilar set of Z . Thus $\text{card}(X) = n_0$. \square

THEOREM 6.2. *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic 1-complex SXM with Φ output-distinguishable and input-complete and A_Z minimal, X a canonical domain-dissimilar set of Z , H a partition induced by X on Z and $Z/H = \{Z'\}$. Then Z' is a minimal cover of Z .*

Proof. If n_0 is the number of states of a minimal cover of Z then, using Lemma 6.2 we have $\text{card}(H) = \text{card}(X) = n_0$. Thus it is sufficient to prove that for any deterministic SXM Z'' obtained by removing one or more arcs from Z' , $L_{A_Z} \subseteq L_{A_{Z''}}$ does not hold.

Let $Z'' = (\Sigma, \Gamma, H, M, \Phi, F'', H_0, m_0)$ as above. Then $\exists \phi \in \Phi, q \in Q, H_1 \in H$ such that $(q, \phi) \in \text{dom}(F)$, $(H_1, \phi) \notin \text{dom}(F'')$ and $q \in H_1$. Let $p \in \Phi^*$ such that $F^*(q_0, p) = q$. Then $(q_0, p\phi) \in \text{dom}(F^*)$. If $(H_0, p) \in \text{dom}(F''^*)$ then $F''^*(H_0, p) = H_1$ so $(H_0, p\phi) \notin$

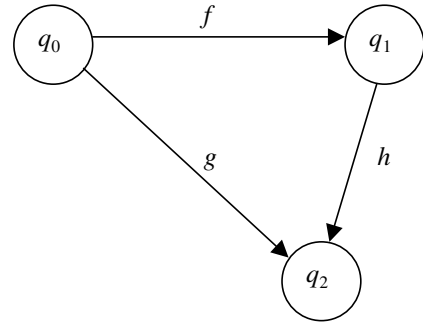


FIGURE 6. The associated FA of Z .

$\text{dom}(F''^*)$. Thus $p\phi \in L_{A_Z} \setminus L_{A_{Z''}}$ so $L_{A_Z} \subseteq L_{A_{Z''}}$ does not hold. \square

Thus, any canonical domain-dissimilar set of Z gives rise to a minimal cover of Z . However, not all minimal covers of Z can be obtained in this manner, as is illustrated by the following counter-example.

EXAMPLE 6.1. Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic SXM with $Q = \{q_0, q_1, q_2\}$, $\Phi = \{f, g, h\}$ with f, g, h output-distinguishable and input-complete and $\text{dom}(f) \cap \text{dom}(g) = \emptyset$, $\text{dom}(f) \cap \text{dom}(h) = \emptyset$ and $\text{dom}(g) \cap \text{dom}(h) \neq \emptyset$ and F as represented in Figure 6. Then q_0 and q_2 are domain-compatible, q_1 and q_2 are domain-compatible but q_0 and q_1 are not domain-compatible.

It can be observed that the only canonical dissimilar set of Z is $X = \{q_0, q_1\}$, which induces partitions H_1 and H_2 , where:

$$\begin{aligned} H_1 &= \{\{q_0\}, \{q_1, q_2\}\}, \\ H_2 &= \{\{q_0, q_2\}, \{q_1\}\}. \end{aligned}$$

On the other hand, there are three admissible and domain-consistent decompositions of Q with minimum number of elements (two). These are H_1 , H_2 and H_3 , where

$$H_3 = \{\{q_0, q_2\}, \{q_1, q_2\}\}.$$

Unlike H_1 and H_2 , H_3 is not a partition.

In order to find all minimal covers of Z we construct $Z/H_1 = \{Z_1\}$, $Z/H_2 = \{Z_2\}$, $Z/H_3 = \{Z_1, Z_2, Z_3, Z_4\}$, where:

$$\begin{aligned} Z_1 &= (\Sigma, \Gamma, H_1, M, \Phi, F_1, \{q_0\}, m_0) \\ &\quad \text{with } F_1 \text{ represented in Figure 7a,} \\ Z_2 &= (\Sigma, \Gamma, H_2, M, \Phi, F_2, \{q_0, q_2\}, m_0) \\ &\quad \text{with } F_2 \text{ represented in Figure 7b,} \\ Z_3 &= (\Sigma, \Gamma, H_3, M, \Phi, F_3, \{q_0, q_2\}, m_0) \\ &\quad \text{with } F_3 \text{ represented in Figure 7c,} \\ Z_4 &= (\Sigma, \Gamma, H_3, M, \Phi, F_4, \{q_0, q_2\}, m_0) \\ &\quad \text{with } F_4 \text{ represented in Figure 7d.} \end{aligned}$$

It can be easily observed that Z_1 , Z_2 , Z_3 and Z_4 are all minimal covers of Z .

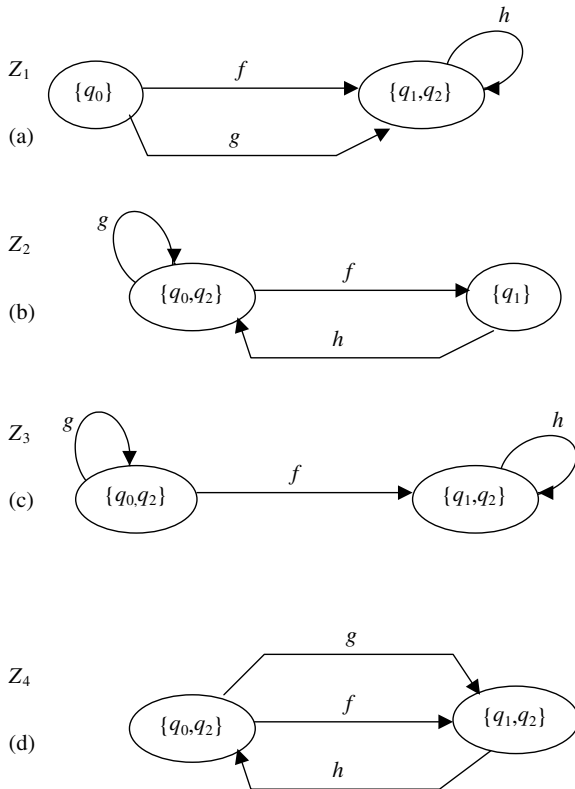


FIGURE 7. The associated FAs of Z_1 (a), Z_2 (b), Z_3 (c) and Z_4 (d).

7. CONCLUSIONS

This paper has investigated the minimality issue in the context of deterministic stream X-machines and considered two types of minimality: *state-minimal* stream X-machine with respect to Φ and *minimal cover* with respect to Φ .

A state-minimal stream X-machine with respect to Φ is the ‘smallest’ (in terms of states and arcs) deterministic stream X-machine Z with type Φ that computes the function f_Z . The paper has identified necessary conditions for an SXM to be state-minimal. It has also been shown that if the type of the SXM satisfies the two ‘design for test conditions’ (output-distinguishability and input-completeness) then state-minimality is equivalent to the minimality of the associated automaton.

A minimal cover with respect to Φ of a given deterministic stream X-machine is the ‘smallest’ deterministic stream X-machine with type Φ that computes a function that includes that of the original machine. This paper has provided an algorithm for constructing all minimal covers of deterministic stream X-machines with type input-complete and output-distinguishable. The construction of a minimal cover for a 1-complex deterministic stream X-machine with type input-complete and output-distinguishable has been investigated further and it has been shown that any canonical domain-dissimilar set of the machine induces a minimal cover.

The results above can be slightly generalized in the sense that the ‘design for test conditions’ can be replaced by the following, more general, variants.

DEFINITION 7.1. Let $U = \{U_\phi \mid \phi \in \Phi\}$ with $U_\phi \subseteq \Sigma$ and $U_\phi \neq \emptyset \forall \phi \in \Phi$. Then Φ is called *input-complete* with respect to U if $\forall \phi \in \Phi$ and $\forall m \in M \exists \sigma \in U_\phi$ such that $(m, \sigma) \in \text{dom}(\phi)$.

DEFINITION 7.2. Let $U = \{U_\phi \mid \phi \in \Phi\}$ with $U_\phi \subseteq \Sigma$ and $U_\phi \neq \emptyset \forall \phi \in \Phi$. Then Φ is called *output-distinguishable* with respect to U if $\forall \phi_1, \phi_2 \in \Phi$ if $\exists m, m_1, m_2 \in M, \sigma \in U_{\phi_1} \cup U_{\phi_2}, \gamma \in \Gamma$ with $\phi(m, \sigma) = (\gamma, m_1)$ and $\phi_2(m, \sigma) = (\gamma, m_2)$ then $\phi_1 = \phi_2$.

It is not difficult to check that none of the results in this paper are affected by this replacement.

The concepts presented in this paper have important implications for practical analysis and testing of software systems. The construction of minimal models for a system is a common scenario in an object-oriented environment. It is also particularly useful for the application of the stream X-machine based testing method if the processing functions are routines from a library or are existing pieces of software that have been shown to be correct.

Other types of minimal stream X-machines, such that the minimal stream X-machine that covers a function whose domain contains sequences of a limited length, are currently being investigated.

ACKNOWLEDGEMENT

The author is indebted to the anonymous referees who made a number of useful comments on the first version of this paper.

REFERENCES

- [1] Holcombe, M. and Ipate, F. (1998) *Correct Systems: Building a Business Process Solution*. Springer, Berlin.
- [2] Ipate, F. and Holcombe, M. (1996) Another look at computability. *Informatica*, **20**, 359–372.
- [3] Eilenberg, S. (1994) *Automata, Languages and Machines*, Vol. A. Academic Press, New York.
- [4] Holcombe, M. (1988) X-machines as a basis for dynamic system specification. *Software Eng. J.*, **3**, 69–76.
- [5] 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 Electron. Model.*, **3**, 141–161.
- [6] 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 and Engineering Society Press, Athens.
- [7] Kefalas, P. and Kapeti, E. (2000) A design language and tool for X-machine specification. In Fotadis, D. I. and Nikolopoulos, S. D. (eds), *Advances in Informatics*, pp. 134–145. World Scientific, Athens.
- [8] Ipate, F. and Holcombe, M. (1998) A method for refining and testing generalized machine specifications. *Int. J. Comput. Math.*, **68**, 197–219.
- [9] Ipate, F. and Holcombe, M. (2002) An integrated refinement and testing method for stream X-machines. *Appl. Algebra Eng., Commun. Comput.*, **13**, 67–91.

- [10] Bălănescu, T., Gheorghe, M. and Holcombe, M. (2000) Deterministic stream X-machines based on grammar systems. In Martin-Vide, C. and Mitrana, V. (eds), *Words, Sequences, Grammars, Languages: where Biology, Computer Science, Linguistics and Mathematics Meet*, Vol. 1, pp. 13–23. Kluwer, Dordrecht.
- [11] Bălănescu, T., Gheorghe, M., Holcombe, M. and Ipate, F. (2001) Testing collaborative agents defined as stream X-machines. In *Advances in Artificial Life, Proc. 6th Euro. Conf. ECAL*, Prague, Czech Republic, September 10–14, pp. 296–305. Springer, Berlin.
- [12] Gheorghe, M. (2001) Generalized stream X-machines and cooperating distributed grammar systems. *Formal Aspects Comput.*, **12**, 459–472.
- [13] Barnard, J., Whitworth, J. and Woodward, M. (1996) Communicating X-machines. *Inform. Software Technol.*, **38**, 401–407.
- [14] Bălănescu, T., Cowling, T., Georgescu, H., Gheorghe, M., Holcombe M. and Vertan, C. (1999) Communicating stream X-machines are no more than X-machines. *J. Universal Comput. Sci.*, **5**, 494–507.
- [15] Cowling, A., Georgescu, H. and Vertan, C. (2000) A structured way to use channels for communication in X-machine systems. *Formal Aspects Comput.*, **12**, 458–500.
- [16] Georgescu, H. and Vertan, C. (2000) A new approach to communicating X-machines. *J. Universal Comput. Sci.*, **6**, 490–502.
- [17] Ipate, F. and Holcombe, M. (2002) Testing conditions for communicating stream X-machine systems. *Formal Aspects Comput.*, **13**, 431–446.
- [18] Aguado, J., Bălănescu, T., Cowling, T., Gheorghe, M., Holcombe, M. and Ipate, F. (2002) P systems with replicated rewriting and stream X-machines (Eilenberg machines). *Fundamenta Informaticae*, **49**, 17–33.
- [19] Ipate, F. and Holcombe, M. (1997) An integration testing method that is proved to find all faults. *Int. J. Comput. Math.*, **63**, 159–178.
- [20] Holcombe, M., Ipate, F. and Grondoudis, A. (1995) Complete functional testing of safety-critical systems. *Proc. 2nd IFAC Workshop on Safety and Reliability in Emerging Control Technologies*, Daytona Beach, FL, 1–3 November, pp. 199–204. Elsevier, Oxford.
- [21] Ipate, F. and Holcombe, M. (1998) Specification and testing using generalized machines: a presentation and a case study. *Softw. Test. Verif. Reliab.*, **8**, 61–81.
- [22] Cheng, K.-T. and Krishnakumar, A. S. (1993) Automatic functional test generation using the extended finite state machine model. In *Proc. 30th Design Automation Conf.*, Dallas, TX, June 14–18, pp. 86–91. ACM Press, New Orleans.
- [23] Lee, D. and Yannakakis, M. (1996) Principles and methods of testing finite state machines—a survey. *Proc. IEEE*, **84**, 1090–1123.
- [24] Wang, C.-J. and Liu, M. T. (1993) Generating test cases for EFSM with given fault models. In *Proc. IEEE INFOCOM '93*, Vol. 2, March 28–April 1, San Francisco, CA, pp. 774–781. IEEE, San Francisco.
- [25] Ostrand, T. J. and Balcer, M. J. (1989) The category-partition method for specifying and generating functional tests. *Commun. ACM*, **31**, 667–686.
- [26] Ipate, F. and Holcombe, M. (2000) Generating test sequences from non-deterministic generalized stream X-machines. *Formal Aspects Comput.*, **12**, 443–458.
- [27] Bălănescu, T. (2000) Generalized stream X machines with output delimited type. *Formal Aspects Comput.*, **12**, 473–484.
- [28] Hierons, R. M. and Harman, M. (2000) Testing conformance to a quasi-non-deterministic stream X-machine. *Formal Aspects Comput.*, **12**, 423–442.
- [29] Cohen, D. I. A. (1996) *Introduction to Computer Theory* (2nd edn). John Wiley & Sons, New York.
- [30] Campeanu, C., Santean, N. and Yu, S. (2001) Minimal cover-automata for finite languages. *Theor. Comput. Sci.*, **267**, 3–16.
- [31] Campeanu, C., Paun, A. and Yu, S. (2002) An efficient algorithm for constructing minimal cover automata for finite languages. *Int. J. Found. Comput. Sci.*, **13**, 83–97.
- [32] Bogdanov, K., Fairtlough, M., Holcombe, M., Ipate, F. and Jordan, C. (1997) *X-machine Specification and Refinement of Digital Devices*. Memoranda in Computer Science CS-97-16, Department of Computer Science, University of Sheffield.