A TESTING PROCEDURE FOR DETERMINISTIC COVER FINITE STATE MACHINES

Florentin Ipate, Tudor Balanescu

Department of Mathematics and Computer Science
University of Pitesti, Romania
fipate@ifsoft.ro

Abstract The paper deals with the testing procedures of software systems specified by finite sequential functions. It is shown that using the cover finite state machines, the \(W\) and \(Wp\) testing methods may be adapted to cope with such systems.

1. INTRODUCTION

The testing methodologies for software systems modeled by finite state machines are mainly aimed to the automatic generation of a test set such that a correct behaviour of the system on all the cases of the test guarantees the correctness in the general case. Obviously, that goal, contradicting the famous statement by E. W. Dijkstra program testing can be used to show the presence of bugs, but never to show their absence may be accomplished only in some restricted classes of software systems. Such a class comprises the communication protocols specified by finite state machines but implemented by some other finite state machines that meet some additional requirements concerning their form and size.

A special case is the test generating procedure for protocols modeled by deterministic finite state machines using the so called \(W\)–method [2]. Given two deterministic finite state machines \(S\) and \(I\), the former representing the specification and the latter the implementation, a test set is a set of sequences that, when applied to the two machines with identical results, guarantees their identical behaviour (if the two machines are not equivalent then at least one of these sequences will produce different results on the two machines). The test set will be generated from the specification \(S\) and, in principle, no information is available about the implementation \(I\), except that the difference between the number of states of the implementation and that of the specification has to be at most \(k\), a positive integer estimated by the tester. A variant of this method, called the \(Wp\)-method, that, in certain circumstances may slightly reduce the size of the test set at the expense of the complexity of the generation algorithm, has also been developed [7]. The method was also generalized to non-deterministic finite state machines [17]. The case of partially specified de-
terministic finite state machines is also considered in [1]. Furthermore, the \( W \) method has been used as a basis for test set generation for more complex models such as a type of extended finite state machine called \textit{stream X-machine} (Eilenberg machine) [9]; it has been shown that, in certain circumstances, generating test sets for stream X-machines (Eilenberg machines) can be reduced to generating test sets for deterministic finite state machines that are partially specified [10], [9], [12], [11], [14], [13].

Due to the fact that many applications of regular languages use actually only finite languages, some recent investigations on regular languages and finite automata [4], [20] are focused on the specification of finite languages. As the number of states of the automaton that accepts a finite language is at least one more than the length of the longest word in the language, and can even be in the order of exponential to that number, these papers deal with the so called \textit{cover finite automata}. Informally, a cover finite automaton \( A \) of a finite language \( L \) is a finite automaton that accepts all words in \( L \) and possibly other words that are strictly longer than any word in \( L \). In many cases, a minimal deterministic cover automaton of \( L \) has a much smaller size than a minimal deterministic automaton that accepts \( L \). Thus, in practice cover automata can be used to reduce the size of automata for finite languages.

In this paper we adapt the \( W \) and \( Wp \) methods to work with \textit{cover finite state machines} taking advantage of the reduced complexity of the specifications based on these tools.

2. \textbf{FINITE STATE MACHINES}

This section introduces the finite state machine and related concepts and results that will be used later in the paper.

For an alphabet \( A \), and a set \( U \subseteq A^* \), we define \( U^n \) by \( U^0 = \{ \epsilon \} \) and \( U^n = U^{n-1}U \) for \( n \geq 1 \). Also, \( U[n] = \cup_{0 \leq k \leq n} U^k \). 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.

\textbf{Definition 2.1.} A deterministic finite state machine (DFSM for short) \( M \) is a tuple \( (\Sigma, \Gamma, Q, h, q_0) \), as follows:

- \( \Sigma \) is the finite input alphabet;
- \( \Gamma \) is the finite output alphabet;
- \( Q \) is the finite set of states;
- \( h \) is the (partial) next state and output function, \( h : Q \times \Sigma \rightarrow Q \times \Gamma \);
A Testing Procedure for Deterministic Cover Finite State Machines

$q_0$ is the initial state $q_0 \in Q$.

A DFSM is usually described by a state transition diagram. $M$ is said to be completely specified if $h$ is a total function. Otherwise $M$ is said to be partially specified. Moreover, the (partial) function $h : Q \times \Sigma \rightarrow Q \times \Gamma$ may be seen as broken up into two (partial) functions:

- $h_1 : Q \times \Sigma \rightarrow Q$, the next state function;
- $h_2 : Q \times \Sigma \rightarrow \Gamma$, the output function

such that $h(q, \sigma) = (h_1(q, \sigma), h_2(q, \sigma))$.

**Definition 2.2.** The next state function $h_1$ can be extended to a (partial) function $h_1^* : Q \times \Sigma^* \rightarrow Q$ defined by

- $h_1^*(q, \epsilon) = q$, $q \in Q$;
- $h_1^*(q, s\sigma) = h_1(h_1^*(q, s), \sigma)$, $q \in Q, s \in \Sigma^*, \sigma \in \Sigma$.

The output function $h_2$ can be extended to a (partial) function $h_2^* : Q \times \Sigma^* \rightarrow \Gamma^*$ defined by

- $h_2^*(q, \epsilon) = \epsilon$, $q \in Q$;
- $h_2^*(q, s\sigma) = h_2(q, s)h_2(h_1^*(q, s), \sigma)$, $q \in Q, s \in \Sigma^*, \sigma \in \Sigma$.

**Definition 2.3.** For $q \in Q$, the function computed by $M$ in $q$, denoted by $\{q\}_M$, is defined by

$$\{q\}_M(s) = h_2^*(q_0, s), s \in \Sigma^*.$$  

The function computed by $M$ in $q_0$ is simply called the function computed by $M$ and is denoted by $\{M\}$.

**Definition 2.4.** A state $q \in Q$ is called accessible if $\exists s \in \Sigma^*$ such that $h_1^*(q_0, s) = q$. $A$ is called accessible if $\forall q \in Q$, $q$ is accessible.

**Definition 2.5.** Two states $q_1, q_2 \in Q$ are called $Y$-equivalent, $Y \subseteq \Sigma^*$, if $h_2(q_1, s) = h_2(q_2, s) \forall s \in Y$. Otherwise $q_1$ and $q_2$ are called $Y$-distinguishable. If $Y = \Sigma^*$ then $q$ and $q'$ are simply called equivalent or distinguishable. Two DFSMs are called ($Y$-)equivalent or ($Y$-)distinguishable if their initial states are ($Y$-)equivalent or ($Y$-)distinguishable.

**Definition 2.6.** $M$ 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.** $M$ is called minimal if any other DFSM that computes $\{M\}$ has at least the same number of states as $M$. 
Theorem 2.1. $M$ is minimal if and only if $M$ is accessible and reduced.

This is a well known result, for a proof see for example [6].

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

- $g$ is bijective;
- $g(q_0) = q'_0$;
- $g(h_1(q, \sigma)) = h'_1(g(q), \sigma), \forall q \in Q, \sigma \in \Sigma$;
- $h_2(q, \sigma) = h'_2(g(q), \sigma), \forall q \in Q, \sigma \in \Sigma$.

Theorem 2.2. For two minimal DFSMs $M$ and $M'$, $\{M\} = \{M'\}$ if and only if $M$ and $M'$ are isomorphic.

This is a well known result, for a proof see for example [6]. Techniques for constructing the minimal DFSM that computes the same function as a given DFSM also exist, for more detail see for example [6] or [3].

Definition 2.9. A (partial) function $F : \Sigma^* \rightarrow \Gamma^*$ is called a sequential function (S-function for short) if $F = \{M\}$ for some DFSM $M$.

Definition 2.10. Given an S-function $F : \Sigma^* \rightarrow \Gamma^*$ and $s \in \text{dom } F$, we denote by $F^s : \Sigma^* \rightarrow \Gamma^*$ the (partial) function defined by: $F^s(x) = F(s)^{-1}F(sx), x \in \Sigma^*$ such that $sx \in \text{dom } F$.

Naturally, $F^s$ is also an S-function. Furthermore, it is easy to verify that if $M = (\Sigma, \Gamma, Q, h, q_0)$ is such that $F = \{M\}$, then $F^s = \{M\}$, where $q = h_1(q, s)$.

Definition 2.11. A sequential function $F : \Sigma^* \rightarrow \Gamma^*$ is called a finite sequential function (FS-function for short) of length $l$ if dom $F$ is finite and $l$ is the length of the longest word(s) in dom $F$.

Obviously, if $F$ is an FS-function, then $F^s$ is also an FS-function.

3. THE $W$ AND $WP$ METHODS FOR DFSMS

Here we present the basis of the methodology used for automatic generation of test sequences from a DFSM specification.

3.1. THE $W$-METHOD

The $W$- method involves the selection of two sets of input sequences, a state cover and a characterisation set, as defined next.
Definition 3.1. \( S \subseteq \Sigma^* \) is called a \textit{state cover} of \( M = (\Sigma, \Gamma, Q, h, q_0) \) if \( \epsilon \in S \) and \( \forall q \in Q \setminus \{q_0\}, \exists s \in S \) such that \( h^*_1(q_0, s) = q \).

Definition 3.2. \( W \subseteq \Sigma^* \) is called a \textit{characterisation set} of \( M = (\Sigma, \Gamma, Q, h, q_0) \) if any two distinct states of \( M, q, q' \in Q, q \neq q' \), are \( W \)-distinguishable.

Note that a state cover and a characterisation set exist if \( M \) is minimal.

The test set generated by the \( W \)-method in the context of completely specified DFSMs is

\[
U_k = S\Sigma[k + 1]W,
\]

where

- \( S \) is a state cover of the specification \( M \)
- \( W \) is a characterisation set of the specification \( M \)

The idea is that the set \( S\Sigma[1] \) (usually called a \textit{transition cover} of \( M \)) ensures that all the states and all the transitions of \( M \) are also present in \( M' \) and \( \Sigma[k]W \) ensures that \( M' \) is in the same state as \( M \) after each transition is used. Notice that the latter set contains \( W \) and also all sets \( \Sigma[i]W \), \( 1 \leq i \leq k \).

This ensures that \( M' \) does not contain extra states. If there were up to \( k \) extra states, then each of them would be reached by some input sequence of up to length \( k \) from the existing states.

Theorem 3.1. [2] Let \( M = (\Sigma, \Gamma, Q, h, q_0) \) and \( M' = (\Sigma, \Gamma, Q', h', q_0') \) be completely specified DFSMs, \( M \) minimal, such that \( \text{card}(Q') - \text{card}(Q) \leq k, k \geq 0 \). Then \( M \) and \( M' \) are equivalent if and only if \( M \) and \( M' \) are \( U_k \)-equivalent.

However, the \( W \)-method in the above form does not work for partially specified finite state machines. Intuitively, this happens because, if \( M \) or \( M' \) are not completely specified, \( M \) and \( M' \) may be \( \{s\}\)-equivalent for a word \( s \) but \( \{t\}\)-distinguishable for some prefix \( t \) of \( s \). Therefore a solution to the problem would be to take the set of all prefixes of \( U_k \) instead of just \( U_k \). In [1] it is shown that only a subset of this set is required, this is

\[
U'_k = U_k \cup S\Sigma[k]\Sigma = S\Sigma[k + 1]W \cup S\Sigma[k]\Sigma.
\]

Theorem 3.2. [1] Let \( M = (\Sigma, \Gamma, Q, h, q_0) \) and \( M' = (\Sigma, \Gamma, Q', h', q_0') \) be (possibly partially specified DFSMs), \( M \) minimal, such that \( \text{card}(Q') - \text{card}(Q) \leq k, k \geq 0 \). Then \( M \) and \( M' \) are equivalent if and only if \( M \) and \( M' \) are \( U'_k \)-equivalent.
3.2. THE WP-METHOD

A variant of the W-method is the partial W method (Wp-method) \[7\]. This reduces the size of the test set at the expense of a slightly more complex generation algorithm. Instead of using the whole set W to check each state \(q\), only a subset of this set can be used in certain cases. This subset \(W_q\) depends on the reached state \(q\) and is called an identification set of \(q\).

**Definition 3.3.** For \(q \in Q\), \(W_q \subseteq \Sigma^*\) is called an identification set of \(q\) if \(\forall q' \in Q \setminus \{q\}, q\) and \(q'\) are \(W_q\)-distinguishable.

**Definition 3.4.** A set \(W \subseteq 2^{\Sigma^*}\) that contains an identification set \(W_q\) of \(q\) for each state \(q\) of \(M\) is called an identification set of \(M\).

Naturally, the union of the identification sets \(W_q \in W\) is a characterization set.

For completely specified DFSMs, the test set generated by the Wp-method is

\[ V_k = S\Sigma[k]W \cup R\Sigma[k] \otimes W, \]

where

- \(S\) is a state cover of the specification \(M\)
- \(R = S\Sigma \setminus S\)
- \(W\) is a characterization set of the specification \(M\)
- \(W\) is an identification set of \(M\) such that for each identification sets \(W_q \in W, W_q \subseteq W\).

and for a set \(A \subseteq \Sigma: A \otimes W\) consists of the sequences \(s\) in \(A\) concatenated with the corresponding \(W_q\) such that \(q\) is reached by \(s\), i.e.

\[ A \otimes W = \{st \mid s \in A \land t \in W_q \land h_1(q_0, s) = q \text{ for some } q \in Q\}. \]

Intuitively, the first component \(V'_k = S\Sigma[k]W\) checks that all the states defined by the specification are identifiable in the implementation. At the same time, the transitions leading from the initial state to these states are checked for correct output and state transfer. The second component \(V'_k = R\Sigma[k] \otimes W\) checks the implementation for all the transitions that are not checked by \(V'_k\).

Naturally, since in general the sets \(W_q\) may be proper subsets of \(W\), the \(Wp\) may yield shorter test sets that the \(W\) method.

**Theorem 3.3.** \[7\]

Analogously to the W-method, the Wp-method can also be extended to cope with partially-specified DFSMs \[1\]. In this case, the test set generated is

\[ V'_k = V_k \cup S\Sigma[k]\Sigma = S\Sigma[k]W \cup R\Sigma[k] \otimes W \cup S\Sigma[k]\Sigma. \]
4. DETERMINISTIC COVER FINITE STATE MACHINES OF FS FUNCTIONS

This section introduces the concept of (minimal) deterministic cover finite state machine of a finite sequential function, similar to the (minimal) deterministic finite cover automaton of a finite language [4]. The concepts and results in this section are natural extensions of the concepts and results in [4]. The proofs are similar to those for the corresponding results given in [4] and consequently will be omitted.

A deterministic cover finite state machine of a finite sequential function $F$ is a DFSM that provides the computation specified by $F$ but may also process other words that are longer than any word in $\text{dom} F$.

**Definition 4.1.** Let $M = (\Sigma, \Gamma, Q, h, q_0)$ be a DFSM, $F : \Sigma \rightarrow \Gamma$ an FS-function and $l = \text{length}(\text{dom} f)$. Then $M$ is called a deterministic cover finite state machine (DCFSM for short) of $F$ if $\{M | \Sigma[l] = F\}$.

**Example 4.1.** Consider $F_n : \{a, b\}^* \rightarrow \{0, 1\}^*$, $n \geq 1$, with $F_n(s) = 1^{\text{length}(s)}$, $s \in \Sigma[n]$, $F_n(s) = 1^n0$, $s \in \Sigma^{n+1}$ and undefined elsewhere. Then the 3 DFSM3 represented in Figure 1 (a), (b) and (c) are DCFSMs of $F$.

**Definition 4.2.** A DCFSM $M$ of an FS-function $F$ is called minimal if for any DCFSM $M'$ of $F$, the number of states of $M$ is less or equal to the number of states of $M'$.

The DFSM in Figure 1 (a) is not a minimal DCFSM of $F_n$ defined in Example 4.1 since there are DCFSMs of $F$ (represented in Figure 1 (b) and (c)) with less states than it.

Two similarity relations are defined in [4] and used to characterize and construct a minimal deterministic finite cover automaton of a finite language. These are naturally extended to DFSMs as follows:

- a similarity relation between input words w.r.t. a FS-function $F$;
- an $l$-similarity relation between the states of a DFSM, where $l$ is the length of the longest word(s) in $\text{dom} F$.

Their formal definitions are now presented.

**Definition 4.3.** Let $F : \Sigma^* \rightarrow \Gamma^*$ be a FS-function of length $l$. Then $\sim_F$ is a relation on $\Sigma^*$ defined by: $s \sim_F t$ if $F^s|_{\Sigma[n]} = F^t|_{\Sigma[n]}$, where $n = l - \max\{\text{length}(s), \text{length}(t)\}$. We say that $s$ is similar to $t$ w.r.t. $F$. The relation $\sim_F$ is called the similarity relation on $\Sigma^*$ w.r.t. $F$. When $s \sim_F t$ does not hold we write $s \not\sim_F t$.

**Remark 4.1.** The similarity relation w.r.t. $F$ is reflexive and symmetric but not transitive. For instance, for $F_n$ as defined in Example 4.1 with $n \geq 1$, $\epsilon \sim_F a^{n+1}$, $a \sim_F a^{n+1}$, but $\epsilon \not\sim_F a$. 
Definition 4.1. Let $M = (\Sigma, \Gamma, Q, h, q_0)$ be an accessible DFSM. For each state $q \in Q$ we define $\text{level}_M(q)$ as the length of the shortest path(s) from $q_0$ to $q$, i.e.

$$\text{level}_M(q) = \min\{\text{length}(s) \mid s \in \Sigma^*, h_1^*(q_0, s) = q\}.$$ 

For $M$ as represented in Figure 1 (c) we have $\text{level}_M(i) = i$, $0 \leq i \leq n$.

Definition 4.5. Let $M = (\Sigma, \Gamma, Q, h, q_0)$ be an accessible DFSM. For each state $q \in Q$ we define $x_M(q)$ as the minimum path from $q_0$ to $q$, i.e.

$$x_M(q) = \min\{s \mid s \in \Sigma^*, h_1(q_0, s) = q\},$$

where the minimum is taken according to the quasi-lexicographical order on $\Sigma^*$.

Remark 4.2. If $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is an ordered set, $n > 0$, then the quasi-lexicographical order on $\Sigma^*$, denoted $\prec$, is defined by: $x \prec y$ 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 $M$ as represented in Figure 1 (c) we have $x_M(i) = a_i^i$, $0 \leq i \leq n$.

Definition 4.6. Let $M = (\Sigma, \Gamma, Q, h, q_0)$ be an accessible DFSM and $l \geq 0$. Then $\sim^l_M$ is a relation on $Q$ defined by

$$p \sim^l_M q \text{ if } F^q | \Sigma[n] = F^p|\Sigma[n], \text{ where } n = l - \max\{\text{level}_M(p), \text{level}_M(q)\}.$$ 

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

Remark 4.3. Analogously to the similarity relation on $\Sigma^*$ w.r.t. $F$, the $l$-similarity relation on $Q$ w.r.t. $M$ is reflexive and symmetric but not transitive. For $M_n$ as represented in Figure 1 (a) and $l = n + 1$ with $n \geq 1$, $0 \sim^l_M n + 1$, $1 \sim^l_M n + 1$, but $0 \not\sim^l_M n$.

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

Lemma 4.1. Let $F : \Sigma^* \rightarrow \Gamma^*$ be an FS-function, $l = \text{length}(\text{dom } F)$ and $M = (\Sigma, \Gamma, Q, h, q_0)$ an accessible DCFSM of $F$. Then $\forall p, q \in Q, p \sim^l_M q$ if and only if $x_M(p) \sim^{\text{dom } F} x_M(q)$.

Definition 4.7. An accessible DFSM $M = (\Sigma, \Gamma, Q, h, q_0)$ is called $l$-reduced if $\forall p, q \in Q$ with $p \neq q$, $p \not\sim^l_M q$.

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

The following theorem identifies necessary conditions for a DCFSM to be minimal.
Theorem 4.1. If \( M = (\Sigma, \Gamma, Q, h, q_0) \) is a minimal DCFSM of an FS-function \( F : \Sigma^* \rightarrow \Gamma^* \) then \( M \) is accessible and \( l \)-reduced, where \( l = \text{length} (\text{dom} \ F) \).

Corollary 4.1. A minimal DCFSM of an FS-function \( F : \Sigma^* \rightarrow \Gamma^* \) is also a minimal DFSM.

The converse is, however, false as illustrated by Example 4.1: the DFSM represented in Figure 1 (a) is a minimal DFSM but not a minimal DCFSM.

5. THE \( W \) AND \( Wp \) METHODS FOR BOUNDED WORDS

Now we will extend the \( W \) and \( Wp \) methods to DCFSMs. Given a specification, in the form of a DCFSM \( M \) of an FS-function \( F \), we need to construct a set of input sequences whose length does not exceed \( l \), the length of the longest sequence in \( \text{dom} \ F \) that, when applied to any implementation \( M' \) will detect any response to input sequences of length at most \( l \) that does not conform to the response specified by \( M \), provided that the difference between the number of states of the implementation and that of the specification is at most a nonnegative integer \( k \).

As shown by the next example, this extension is not straightforward, since it is not sufficient to extract the sequences of length at most \( l \) from the test sets that establish equivalence for unbounded sequences.

Example 5.1. For \( M \) represented in Figure 1 (c), \( S = \{ \epsilon, a, \ldots, a^n \} \) is a state cover of \( M \) and \( W = \{ a^{n+1} \} \) is a characterisation set, so \( U_0 = S \Sigma[1] W = \{ \epsilon, a, \ldots, a^n \} \{ \epsilon, a, b \} \{ a^{n+1} \} \) and \( U_0 \cap \Sigma^{n+1} = \{ a^{n+1} \} \). Consider \( M' \) as represented in Figure 1 (d). It is easy to see that \( M \) and \( M' \) are only distinguished by sequences of the form \( bx, x \in \Sigma^n \), so \( M \) and \( M' \) are \( (U_0 \cap \Sigma^{n+1}) \)-equivalent even though \( M' \) is not a DCFSM of \( F_n \). Furthermore, \( U_0' = U_0 \cup S \Sigma, \) and \( U_0' \cap \Sigma^{n+1} = \{ \epsilon, a, \ldots, a^n \} \{ a, b \} \), so \( M \) and \( M' \) are also \( (U_0 \cap \Sigma^{n+1}) \)-equivalent.

We show now how the definitions of the test set generated by \( W \) and \( Wp \) methods can be revised so that these methods can be extended to DCFSMs. These will be called the \( W \) and \( Wp \) methods for bounded words or, for short, the bounded \( W \) and \( Wp \) methods.

5.1. THE \( W \)-METHOD FOR BOUNDED WORDS

The \( W \)-method for bounded words will only select state covers and characterisation sets with special properties, as defined next.

Definition 5.1. \( S \subseteq \Sigma^* \) is called a proper state cover of \( M = (\Sigma, \Gamma, Q, h, q_0) \) if \( \forall q \in Q, \exists s \in S \) such that \( h_1(q_0, s) = q \) and \( \text{length}(s) = \text{level}_M(q) \).
That is, a proper state cover contains sequences of minimum length that reach the states of $M$. In particular, $S = \{x_M(q) \mid q \in Q\}$ is a proper state cover of $M$.

**Definition 5.2.** $W \subseteq \Sigma^*$ is called a strong characterization set of $M = (\Sigma, \Gamma, Q, h, q_0)$ if any for two states of $M$, $q, q' \in Q$ and $k > 0$, $q$ and $q'$ are $\Sigma[k-1]$-equivalent and $\Sigma^k$-distinguishable $\implies q$ and $q'$ are $(W \cap \Sigma^k)$-distinguishable.

That is, a strong characterization set contains sequences of minimum length that distinguish between the states of $M$.

Then the test set generated by the bounded $W$-method in the case where $M$ and $M'$ are completely specified is

$$Y_k = S \Sigma[k+1]W \cap \Sigma[l],$$

where

- $S$ is a proper state cover of $M$
- $W$ is a strong characterization set of $M$

**Example 5.2.** For $M$ represented in Figure 1 (c), $S = \{\varepsilon, a, \ldots, a^n\}$ is a proper state cover of $M$. For $0 \leq i < j \leq n$, $i$ and $j$ are $\Sigma^{n-j}$-equivalent and $\Sigma^{n-j+1}$-distinguishable, so $W = \{a, \ldots, a^n\}$ is a strong characterization set of $M$. Thus $Y_0 = S \Sigma[1]W \cap \Sigma[n+1] = \{\varepsilon, a, \ldots, a^n\}\{\varepsilon, a, b\}\{a, \ldots, a^n\} \cap \Sigma[n+1] = \{a^i b^j a^k \mid 0 \leq i \leq n, 0 \leq j \leq 1, 0 \leq k \leq n, i+j+k \leq n+1\}$. For $M'$ as represented in Figure 1 (d), since $ba^n \in Y_0$, $Y_0$ distinguishes between $M$ and $M'$, these are $Y_0$-distinguishable.

As for unbounded sequences, the above test set may not distinguish between partially specified machines. Similarly, the test set can be extended to the general case where the machines may be partially specified. The extended test set is

$$Y'_k = Y_k \cup S \Sigma[k] \Sigma \cap \Sigma[l].$$

### 5.2. The $W_p$-Method for Bounded Words

Similarly, the $W_p$-method for bounded words selects only those identification sets that contain sequences of minimal length that distinguish the states of the machine. These are called strong identifications sets.

**Definition 5.3.** For $q \in Q$, $W_q \subseteq \Sigma^*$ is called a strong identification set of $q$, if for any state $q' \in Q$ and $k > 0$, $q$ and $q'$ are $\Sigma[k-1]$-equivalent and $\Sigma^k$-distinguishable $\implies q$ and $q'$ are $(W \cap \Sigma^k)$-distinguishable.

A set $W$ that contains a strong identification set $W_q$ of $q$ for each state of $M$ is called a strong identification set of $M$. 


Then the bounded $W_p$-method generates the following test set for the case where the specification and the implementation are completely specified

$$Z_k = S\Sigma[k]W \cap \Sigma[l] \cup R\Sigma[k] \otimes W \cap \Sigma[l],$$

where

- $S$ is a proper state cover of the specification $M$
- $R = S\Sigma \setminus S$
- $W$ is a strong characterisation set of the specification $M$
- $W$ is a strong identification set of $M$ such that for each identification set $W_q \in W, W_q \subseteq W$.

**Example 5.3.** Consider $M, M', S$ and $S'$ as in Example 5.2. Then $R = S\Sigma \setminus S = \{b, \ldots, a^nb\} \cup a^{n+1}$. It is easy to verify that $W_0 = \{a, a^n\}$ is a strong identification set of 0 and for $1 \leq i \leq n$, $W_i = \{a, a^{n-i+1}\}$ is a strong identification set of $i$, so $W = \{\{a\}, \ldots, \{a, \ldots, a^n\}\}$ is a strong identification set of $M$. Thus $Z_0 = SW \cap \Sigma[n+1] \cup R \otimes W \cap \Sigma[n+1] = \{a, \ldots, a^{n+1}\} \cup \{b\} \{a, \ldots, a^n\} \cup \ldots \{a^{n-1}b\} \{a\}$. Since $ba^n \in Z_0$, $Z_0$ distinguishes between $M$ and $M'$.

In the case where the machines may be partially specified, the test set will be extended to

$$Z'_k = Z_k \cup S\Sigma[k]\Sigma \cap \Sigma[l].$$

6. **CONCLUSIONS**

The use of deterministic cover finite state machines instead of finite state machines in case of finite sequential function is a matter of practical interest because it leads to small sized specification of software systems. Apart from the topics concerning the applicability of $W$ and $W_p$ methods in such environments, the theoretical support presented in the paper may be used as a tool for the proof of the error detection power of these methodologies and will be the subject of a forthcoming paper.

**References**


